UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aprendizaje de máquina automático: Un clasificador de clasificadores basado en metacaracterísticas

Autor: José del Castillo Izquierdo
Tutor: Eduardo C. Garrido Merchán
Ponente: Daniel Hernández Lobato

julio 2020

José del Castillo Izquierdo
Aprendizaje de máquina automático: Un clasificador de clasificadores basado en metacaracterísticas

José del Castillo Izquierdo

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi familia y mis amigos.

Resumen

AutoML es uno de los campos más importantes de Machine Learning cuyo propósito es la automatización de la creación de modelos sin la necesidad de usar un conocimiento experto dentro de este ámbito. Opcionalmente, algunas soluciones de AutoML aspiran a conseguir mejores resultados que los modelos de Machine Learning tradicionales (tanto en rendimiento del modelo como en tiempo del proceso).

Con el fin de reducir considerablemente el número de pruebas necesarias para encontrar los mejores algoritmos en problemas de clasificación y regresión, uno de los enfoques principales de AutoML es la selección automática de modelos.

En este trabajo proponemos un selector de clasificadores que predecirá el mejor clasificador, de entre un conjunto de clasificadores candidatos, a partir de las metacaracterísticas de un dataset. Este modelo se entrará a partir de un conjunto de datasets como si fuera un problema de clasificación. Para la realización de esta tarea, probaremos algunos clasificadores evaluándolos bajo diferentes condiciones en el entrenamiento limitando así tanto el número de metacaracterísticas como el número de datasets utilizado. En adición, se estudiarán los efectos que provocan la normalización de los datos en el entrenamiento para cada uno de estos clasificadores de clasificadores.

Partiendo de esa base, presentamos, por otra parte, un clasificador que realizará sus predicciones en base a una combinación eficiente de un conjunto de clasificadores. Para ello, nuestro combinador de clasificadores entrenará un regresor por cada uno de estos clasificadores para estimar la precisión que tendría cada uno de ellos a partir de las metacaracterísticas de un dataset. Se compararán los resultados de nuestro combinador con los modelos que se han utilizado en su combinación para comprobar el rendimiento de nuestro modelo. Del mismo modo, se compararán los resultados de nuestro combinador con un combinador uniforme para comprobar que tan buena es la generación de su política de importancias.

Palabras clave

AutoML, ML, Metacaracterística, Clasificador

Abstract

AutoML is one of the most important fields in Machine Learning, whose purpose is to automate the creation of models without the need of using expert knowledge in this area. Optionally, some AutoML solutions aim to achieve better results than traditional Maching Learning models (both in model performance and process time).

In order to greatly reduce the number of tests required to find the best algorithms for classification and regression problems, one of AutoML's main approaches is automatic model selection.

In this work we propose a classifier selector that will predict the best classifier, from a set of candidate classifiers, based on the metafeatures of a dataset. This model will be entered from a set of datasets as if it were a classification problem. To carry out this task, we will test some classifiers evaluating them under different training conditions, thus limiting both the number of metafeatures and the number of datasets used. In addition, the induced effects of the normalization of the data in training will be studied for each of these classifiers of classifiers.

Starting from this basis, we present, on the other hand, a classifier that will make its predictions based on an efficient combination of a set of classifiers. To do this, our classifier combiner will train a regressor for each of these classifiers to estimate the precision that each of them would have from the metafeatures of a dataset. The results of our combiner will be compared with the models that have been used in its combination to check the performance of our model. In the same way, the results of our combiner will be compared with a uniform combiner to verify how good the generation of its importances policy is.

Keywords

AutoML, ML, Metafeature, Classifier

Índice

Listas

Lista de figuras

Lista de tablas

1

Introducción

Desde el inicio de nuestros tiempos, el ser humano ha construido modelos (conscientemente o inconscientemente) para explicar lo que le rodeaba. Estos modelos se han usado para resolver problemas específicos y servirnos de utilidad estructurando el conocimiento [15]. Con el paso del tiempo, se ha automatizado la construcción de estos modelos con la ayuda de la tecnología y se han logrado varios hitos en dotar a las máquinas de comportamientos inteligentes con la creación de ingeniosos algoritmos como las redes neuronales [24]. Gracias a esto surge un nuevo campo de investigación *Machine Learning* (ML) , una de las ramas de la Inteligencia Artificial (IA) más investigadas de la actualidad.

Gracias al *Machine Learning* (De ahora en adelante ML) se han desarrollado algoritmos específicos para resolver problemas concretos. La mayoría de estos problemas tienen en común que necesitan datos para realizar predicciones que los resuelvan. Sin embargo, cada problema es diferente y para conseguir buenos resultados necesitamos elegir los modelos que se adapten mejor a la realidad [29], cuyas predicciones sean lo más precisas posibles. A la hora de elegir las herramientas para resolver un problema determinado, utilizar la intuición, la experiencia y el conocimiento teórico puede llegar a ser primordial para llegar a encontrar las soluciones más óptimas. Por tanto, el uso del *Machine Learning* requiere de un dominio en el campo y un sofisticado trabajo que varía en función de las necesidades requeridas.

Por otra parte, hay problemas que también pueden llegar a ser muy costosos en tiempo por la complejidad de estos algoritmos. Además, si tenemos en cuenta que algunos problemas requieren de un alto coste económico para probar estos modelos, algunos problemas pueden llegar a ser inviables.

Si añadimos esto a que todos los problemas deben ser tratados de manera especial ya que no existe un algoritmo general que los resuelva, surge la necesidad de encontrar un sofisticado modelo que permita adaptarse de manera automática a cada uno de estos. Uno de los campos más importantes que tratan de conseguir este objetivo es *Automated Machine Learning* (AutoML) basándose en la automatización de los procesos de ML para elegir los mejores algoritmos e hiperparámetros [35].

Gracias a AutoML el proceso de buscar las mejores herramientas para resolver un problema determinado se hace de forma automática y no precisa de tanto conocimiento teórico experto. Y, además de

simplificar la construcción de los modelos, en algunos casos consiguen un mejor rendimiento frente a los proyectos de ML convencionales como en NASNet [40] donde se crearon automáticamente redes neuronales que superaron el estado del arte en el problema CIFAR-10 en aquella época. Algunas soluciones de AutoML pueden contar con la suficiente interpretabilidad como para que se pueda entender en muy poco tiempo el criterio por el cual se eligen unos modelos y parámetros frente a otros. Además, también se pueden combinar estos algoritmos para conseguir un modelo combinado que obtenga mejores resultados.

En este proyecto se tratará uno de los enfoques de AutoML para la selección de modelos en función de la naturaleza de los datos. Para ello se usarán métodos de *Feature Engineering* para extraer las estadísticas de un conjunto de datasets y se evaluará el rendimiento de algunos clasificadores por cada uno de estos con el fin de entrenar un clasificador que estime que clasificador usar a partir un conjunto de datos determinado. Con esto podremos obtener una herramienta de selección de clasificadores automática que podrá ser utilizada en cualquier otro proyecto de ML en el futuro. Además, se planteará una forma de construir un modelo de *Ensemble Learning* que utilizará estas características para combinar los clasificadores de forma eficiente con el fin de conseguir mejores resultados.

La memoria está compuesta por diferentes secciones. En el estado del arte (Sección 2) se mostrarán otros trabajos relacionados dentro del campo de AutoML así como un panorama general de este campo. En la sección 3 se cubrirán los aspectos que han influido en el desarrollo del trabajo, así como los objetivos e hipótesis que se han tenido en cuenta a lo largo del proyecto. En la sección 4 se verá el diseño del proyecto tanto técnico como funcional además de la planificación de este. En la sección 5 se verán los detalles de los recursos utilizados para llevarlo a cabo y en la sección 6 veremos los resultados obtenidos tras la realización de los experimentos que se han realizado . Por último, en la sección 7 se concluirá con una breve discusión que enlazará todos los puntos y se presentará una posible dirección donde continuar la investigación.

Notación

- $\mathbb{X}$: Conjunto
- D : Dataset
- $\vec{x}$: Vector de características
- y : Valor objetivo
- C : Clasificador
- M : Metadataset
- R : Regresor

2

Estado del arte

En esta sección describiremos los avances más importantes que se han dado el campo de AutoML junto con una breve explicación de algunas de sus utilidades. A continuación, se presentará un breve resumen de su historia antes de introducirnos a los diferentes aspectos dentro de este campo. Por último, finalizaremos esta sección con una conclusión para enlazar todo lo anterior visto.

Automated Maching Learning hizo su aparición por primera vez en la década de los noventa con el fin de automatizar algunos procesos de ML como la elección de los hiperparámetros mediante Optimización de hiperparámetros (HPO) en [8]. Sin embargo, no es en 2004 cuando realmente se usaron técnicas más eficientes para optimizar los hiperparámetros de las *Support Vector Machine* (SVM) en [5] demostrando que *Guided Search* conseguía mejores resultados que *Grided Search* en menos tiempo. Por otra parte, en ese mismo año se vio en [34] las primeras innovaciones en la selección de automática de los atributos más relevantes pero no será hasta el 2015 cuando se habrá indagado lo suficente en el campo de *Feature Engineering* para hacerlo sin dominio experto en [18].

Cinco años más tarde, se propuso el primer intento de automatización completa de un *Pipeline* de ML en [9] en el que optimizaba cada fase de este al mismo tiempo que los hiperparámetros de cada una de sus fases. Por último, se empezaron a usar diferentes técnicas de HPO y selección de algoritmos utilizando la optimización bayesiana en [3] el 2011 y en [38] el 2013 respectivamente.

Con el paso del tiempo, las investigaciones se centraron en su mayoría en profundizar en cada uno de estos campos. Además, en adición al auge del *Deep Learning* se aumentaron las posibilidades y se abrieron innumerables subramas de investigación que hicieron que se incrementara el interés por AutoML así como el rendimiento frente a los modelos de ML tradicionales.

A continuación se verán algunos de estos avances en los campos de AutoML mencionados anteriormente.

2.1. Generación automática de arquitecturas de *Machine Learning*

Antiguamente, era necesario poseer un conocimiento experto para crear un modelo de ML que se adapte a un problema determinado. Con el tiempo se han visto patrones que han dado buenos resultados en la creación de la arquitectura de estos proyectos. No obstante, sigue habiendo un gran número de posibilidades en la combinación de los algoritmos y sus hiperparámetros.

Por lo tanto, uno de los enfoques principales de investigación en AutoML se basan en la búsqueda automática de la mejor arquitectura de un *Pipeline* de *Machine Learning*. Según como abarquemos dicho problema podemos encontrar dos maneras de clasificar la búsqueda.

2.1.1. Estructuras fijas

Por una parte, el problema se puede abarcar suponiendo una estructura fija en la arquitectura del *Pipeline*. Esta es la solución que adopta muchos de los Frameworks como en Auto-Sklearn [10]. Por lo general, un *Pipeline* de ML de estructura fija está compuesto (como se propone en [21]) por una fase de limpieza de datos, una fase de *Feature Engineering* y por último, la aplicación del modelo de ML. Más adelante explicaremos la importancia de las dos primeras fases tanto en 2.3.1 como en 2.3.2.

Limitar la investigación a una estructura fija reduce considerablemente el espacio de búsqueda y simplifica el problema dando una solución lo suficientemente buena como para resolver la mayoría de estos siempre y cuando se disponga de los datos adecuados. Sin embargo, si se quiere llegar a unos mejores resultados, se requiere de una estructura variable que se adapte al problema en cuestión.

2.1.2. Estructuras variables

Al ser mucho más flexibles que las estructuras fijas, las estructuras variables son mucho más potentes pero a su vez mucho más costosas. La primera vez que se hizo un método de generación automática de estructuras variables fue en 2016 [27] mediante programación genética. En este método se propone una generación del *Pipeline* en forma de árbol donde se aplican operaciones genéticas para la combinación de estos.

Este enfoque llegó también a otros ámbitos de la IA como *Deep Learning*. En este caso, el objetivo es la construcción automática de la arquitectura de una red neuronal convolucional profunda mediante la selección de bloques ya sea mediante una red neuronal recurrente como en el caso de NASNet [40] y pNASNet [25] o mediante algortimos genéticos como en el caso de AmoebaNET [32].

También se puede interpretar una estructura variable a la generación automática de los datos. Esto

puede conseguirse gracias al aprendizaje por refuerzo, un tipo de aprendizaje que ha conseguido tales hazañas como conseguir en muy pocas horas de entrenamiento jugando contra sí mismo la maestría en juegos como el ajedrez, el Go y el shogi (AlphaZero [36]).

2.2. Selección de algoritmos y optimización de hiperparámetros (CASH)

Para conseguir los los mejores resultados en un *Pipeline* de *Machine Learning* es necesario realizar una selección de los mejores algoritmos además de una optimización de sus hiperparámetros (HPO). En AutoML esto se hacía anteriormente seleccionando los algoritmos primero y haciendo HPO después. Sin embargo, en 2013 se formalizó por primera vez el concepto de *Combined Algorithm Selection and Hyperparameter Optimization* (CASH) en [38] el cuál consiste en hacer al mismo tiempo la optimización automática de la selección de algoritmos y sus hiperparámetros.

Dentro de CASH hay innumerables métodos para lograr este objetivo. A continuación se expondrán algunos de los más relevantes.

2.2.1. *Grid Search*

Este enfoque fue el primero en utilizarse en [8] para las primeras soluciones comerciales de AutoML. Consiste en dividir el espacio de búsqueda en regiones cuadrículadas de un tamaño fijo y evaluar todos sus vértices para elegir el mejor conjunto de algoritmos e hiperparámetros. Aunque su implementación y su paralelización sea sencilla [23], *Grid Search* presenta una serie de problemas que hacen que no sea uno de los mejores CASH.

Por un lado, el número de vertices (y por tanto evaluaciones del *Pipeline* de ML) aumenta exponencialmente conforme mayor sea el número de hiperparámetros y algoritmos a comprobar. Por otro lado, *Grid Search* no maneja muy bien los hiperparámetros que dependen de algoritmos u otros hiperparámetros. Por último, este enfoque supone que las evaluaciones contiguas dan resultados más parecidos cuando en algunos casos puede eso no ser cierto. No obstante, se puede reducir parcialmente este problema aplicando *Grid Search* sobre cada región de esta como en [16].

2.2.2. *Random Search*

Otra manera de tratar el problema de CASH es mediante *Random Search* [1]. Consiste en evaluar diferentes configuraciones de hiperparámetros y algoritmos elegidos de manera aleatoria.

Al contrario que *Grid Search*, la dependencia de hiperparámetros se puede tratar con más facilidad.

Además de que la convergencia hacia un buen resultado sea más rápida [2], *Random Search* funciona especialmente bien para los problemas con gran cantidad de mínimos locales.

2.2.3. Optimización mediante modelos secuenciales (SMBO)

También se puede interpretar el problema de CASH como un problema de regresión cuya función depende de los hiperparámetros. Este es el enfoque de la Optimización mediante modelos secuenciales(*Sequential Model-Based Optimization* (SMBO)) [3] cuyo objetivo es reducir la función de pérdidas gracias a modelos de regresión probabilística mediante múltiples iteraciones donde se evalua el modelo y se estima una nueva configuración de hiperparámetros cada vez.

Entre los modelos que se utilizan, cabe destacar la optimización Bayesiana mediante procesos gaussianos [39]. Estos modelos se han usado con bastante freciencia pese a su complejidad de $O(n^3)$ en tiempos de ejecución [39] y a la imposibilidad de manejar hiperparámetros categóricos. No obstante, se han presentado avances como extensiones que permiten a los procesos gaussianos tratar con variables enteras [11].

Por último, existen métodos de regresión basados en *Random Forest* [4] que permiten tratar con los datos categóricos mediante múltiples árboles de decisión. Estos tienen la ventaja de que son más rápidos tanto en su entrenamiento como en la evaluación en comparación a los procesos gaussianos.

2.2.4. Algoritmos evolutivos

Una manera de enfocar la optimización de CASH es el uso de los algoritmos evolutivos [7]. Estos métodos usan comportamientos de la biología evolutiva para resolver problemas y, al ser algoritmos paramétricos, ofrecer cierta flexibilidad para encontrar las mejores configuraciones.

Hay una gran cantidad de métodos dentro de esta rama como los algoritmos genéticos, la programación genética o la optimización de enjambre de partículas (*Particle Swarm Optimization* (PSO)) [33]. Este último se usó en la creación del primer *Pipeline* automático de ML [9] y en otras investigaciones como en [6]. Consiste en una distribución de partículas que exploran por separado diferentes regiones del espacio de búsqueda mediante movimientos aleatorios basados tanto en el mejor resultado de la particula individual como el mejor resultado del enjambre.

2.3. Tratamiento automático de los datos

Antes de utilizar un modelo, el tratamiento de los datos es una parte esencial dentro de un *Pipeline* de ML tal como se vió en la sección 2.1.1 y es necesario para conseguir los mejores resultados

posibles. Es por eso razón que uno de los objetivos de AutoML es automatizar este proceso independientemente del tipo de problema.

2.3.1. Limpieza automática de los datos

En general, a medida que se trabaja con una gran cantidad de datos podemos encontrarnos con algunos problemas [31] como instancias con atributos vacíos, datos invalidos o instancias redundantes. Tanto en *Machine Learning* como en AutoML existe una fase que se encaga de tratar esto, *Data Cleaning*. Sin embargo, pese a que existan algunos *frameworks* que cuenten con este proceso [10], normalmente no es sensible a la optimización del *Pipeline* y, por tanto, muchos *frameworks* actuales prefieren suponer que esta fase es responsabilidad del usuario.

Sin embargo, pese a que aún no hayan muchas investigaciones notables referentes a la limpieza automática de datos, se ha demostrado que puede aumentar drásticamente la calidad de estos [17]. Además, ha resultado ser útil para atender a los requisitos de algunos algoritmos adecuando sus entradas mediante transformaciones de metacaracterísticas [14] y no solo en las fases tempranas del *Pipeline*.

2.3.2. *Feature Engineering*

Una de las fases más importantes de un *Pipeline* de ML es *Feature Engineering*. Este proceso permitará mejorar drásticamente el rendimiento del *Pipeline* [30], seleccionando lo atributos más importantes o generando nuevos atributos a partir de estos. Por tanto, dependiendo de la tarea que se realice, podemos dividir *Feature Engineering* en *Feature Extraction*,*Feature Selection* y *Feature Construction* [26]. Se requiere de conocimiento experto para automatizar *Feature Construction*, por esa razón, las investigaciones de AutoML se han centrado más en los otros tipos.

Feature Extraction

Este tipo de *Feature Engineering* se encargará de generar nuevos atributos a partir de transformaciones de los atributos originales [37]. Estos operadores pueden ser de 3 tipos dependiendo del número de atributos que utiliza los cuales son:

- Unarios: Transformaciones de un atributo como normalizaciones o discretizadores de un atributo.
- Binarios: Tranformaciones de dos atributos como la correlación de dos atributos [20].
- N-arios: Tranformaciones de 3 o más atributos como las estadísticas de una distribución.

Normalmente, la extracción de los atributos se hace aplicando iterativamente estas tranformaciones partiendo de un dataset inicial para generar nuevos atributos. Una vez generados, se añadirán al conjunto de atributos totales para luego ser seleccionados mediante métodos de *Feature Selection*.

Entre los enfoques más relevantes de *Feature Extraction* podemos destacar la interpretación de esta extracción de atributos como un problema de selección de nodos donde se define un árbol de transformaciones [22] en el que los operadores son las ramas y los atributos son los nodos.

Feature Selection

El objetivo de *Feature Selection* es seleccionar un subconjunto de los atributos más relevantes de un conjunto de datos para mejorar el rendimiento de un modelo [26]. Hay muchos métodos para conseguir este objetivo.

Por una parte, se puede utilizar la teoría estadística [28] para aplicar algoritmos como umbral de varianza que consiste en descartar los atributos que cuya varianza total no superen cierto umbral. Por otra parte, se pueden utilizar modelos pre-entrenados para predecir la importancia de cada característica [19]. Por último, también existen otras opciones como aplicar aprendizaje por refuerzo [12] para entrenar un modelo que aprenda de diferentes resultados obtenidos por un subconjuntos de atributos aleatorio del conjunto inicial.

2.4. Conclusiones

Podemos concluir a partir de los trabajos relacionados, que las investigaciones de AutoML se han centrado, en su mayoría, en automatizar una o varias fases de ML sin la aplicación de un conocimiento teórico externo. De esta manera se consiguen estudiar nuevas formas de usar los datos, que ayudarán a abrir nuevas líneas de investigación y mejorar el rendimiento de muchos *Pipelines* futuros. Esto es posible debido a que, al no necesitar un conocimiento teórico externo, el algoritmo de AutoML se puede aplicar a la mayoría de problemas.

Con este propósito, nuestro trabajo se enfocará en la fase de *Model Selection* para definir una posible forma de construir un selector de clasificadores automático que realizará su elecciones en base a un conjunto de datos dado.

3

Definición del proyecto

Una vez visto los anteriores trabajos relacionados se van a presentar los aspectos que han influido al desarrollo de este TFG para definir el alcance de este proyecto.

3.1. Objetivos

En esta subsección veremos los objetivos que se han establecido para llevar a cabo el TFG.

O-1.– Se implementará una herramienta que clasifique que clasificador usar para un nuevo conjunto de datos.

O-2.– Se implementará un combinador de clasificadores que estimará el desempeño de cada clasificador candidato en función del dataset y lo combinará para las evaluaciones de las instancias.

O-3.– Se recolectarán 18 datasets los cuáles, al evaluar cada clasificador candidato con estos, el número de clasificadores que mejores resultados han obtenido frente a los demás sea equilibrado.

O-4.– Nuestro modelo seleccionará las metacaracterísticas más relevantes para predecir el clasificador y entrenar tanto el clasificador de clasificadores como el combinador de clasificadores.

O-5.– Se comparará el rendimiento de diferentes clasificadores para cumplir el objetivo O1.

O-6.– Se compararán los resultados obtenidos con algunos frameworks de AutoML.

3.2. Hipótesis

En esta subsección se presentarán las suposiciones que se han tenido previamente al desarrollo del proyecto. Más adelante en la sección 7 se corroborará la veracidad o falsedad de estos.

H-1.– A partir de la naturaleza de los datos se puede predecir qué modelo se ajusta mejor a un problema.

H-2.– Se pueden combinar los modelos que mejor funcionan para conseguir un mejor rendimiento en la predicción final.

H-3.– Normalizar los datos después de obtener las metacaracterísticas permitirá obtener mejores resultados en los experimentos.

H-4.– Aumentar el número de los conjuntos de datos en el entrenamiento permitirá que nuestro modelo obtenga mejores resultados.

H-5.– Disminuir el número de metacaracterísticas relevantes seleccionadas del metadataset permitirá que nuestro

modelo obtenga mejores resultados.

H-6.– El proceso gaussiano es el mejor modelo para cumplir el objetivo O1 debido a que no precisa de tantos datos para obtener buenos resultados.

3.3. Asunciones

A continuación se listarán un conjunto de asunciones que se han tenido en cuenta para acotar y simplificar el problema y así limitar el alcance del proyecto.

A-1.– Al abarcarar con problemas de clasificación binaria en la extracción de los datasets se asume que si se tomasen otro tipo de problemas entonces los resultados serían similares.

A-2.– Se asume que aunque los experimentos se hayan basado únicamente en la decisión de 3 clasificadores, se podrían haber usado más si se contara con una cantidad mayor de datos.

A-3.– Aunque no se haya tenido en cuenta la optimización de los hiperparámetros en los clasificadores se asume que el desempeño de cada uno de ellos no afectará a los resultados de los experimentos.

A-4.– Se asume que aunque se hayan impuesto restricciones para la extracción de datasets en la subsección 5.3 esto no alterará los resultados de los experimentos.

A-5.– Se asume que si se hubiera utilizado otro método en la selección de metacaracterísticas relevantes no habría variado significativamente los resultados de los experimentos.

3.4. Restricciones

Por último se expondrá una lista de limitaciones ajenas al proyecto pero que influyeron durante la realización del trabajo.

R-1.– El tiempo disponible total para llevar a cabo esta investigación ha sido de 300 horas.

R-2.– El tiempo de ejecución de grandes cantidades de conjunto de datos ha limitado el número de experimentos realizados.

R-3.– El tiempo para la obtención de los mejores datasets para el entrenamiento del clasificador de clasificadores ha limitado el número de datasets extraidos.

R-4.– Debido al COVID-19 no se ha podido contar durante 180 horas con recursos como ordenadores ofrecidos por la facultad o similares para la ejecución de tareas en paralelo.

R-5.– No se ha dispuesto del dinero necesario para el de clusters o CPU externas para llevar a cabo los experimentos realizados.

R-6.– Debido a las limitaciones humanas no se han podido investigar todas las técnicas relacionadas con el campo de AutoML.

4

Diseño del proyecto

Nuestro trabajo se engloba en conseguir 2 metas principalmente. Por una parte, queremos desarrollar una herramienta que prediga que clasificador utilizar para un conjunto de datos a partir de sus metacaracterísticas de entre un grupo de clasificadores candidatos sin necesidad de evaluarlos directamente. El encargado de realizar esta tarea será el **Clasificador de clasificadores**.

Por otro lado, nos interesa también implementar un clasificador que combine los clasificadores candidatos de forma eficiente en función de las metacaracterísticas obtenidas por un determinado dataset. A partir de ahora nos referiremos a este clasificador como **Combinador de Clasificadores**.

En las siguientes subsecciones se presentarán el diseño ténico y funcional del proyecto así como su planificación.

4.1. Diseño funcional

En esta subsección se describirá formalmente el funcionamiento de cada una de las partes para conseguir estos objetivos y así explicar en la siguiente subsección los principales procesos de nuestro modelo.

4.1.1. Dataset

Todos los datos que se han usado en el proyecto se han almacenados en datasets. Definimos un dataset $D = \{(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)\}$ como un conjunto de instancias de tamaño n donde cada instancia i se define como $(\vec{x_i}, y_i)$ donde $\vec{x_i}$ es su vector de características e y_i su respectivo valor objetivo. Se define un vector de características de tamaño m como $\vec{x_i} = \{x_{i1}, \ldots, x_{im}\}$ donde x_{ij} es una característica de la instancia i y perteneciente al atributo j.

Podemos también definir el dataset como $D = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n, \vec{y}\}$ siendo $\vec{a}_j$ el vector de valores de un atributo j e $\vec{y}$ el vector de valores objetivo del dataset D. Por cada atributo a_{ij} del vector de valores de un atributo i puede tener un valor continuo donde $a_{ij} \in \mathbb{R}$ (atributos continuos) o un valor discreto

donde $a_{ij} \in \mathbb{N}$ (atributos nominales o categóricos). Del mismo modo, dependiendo de la tarea del dataset D, el vector de valores objetivos $\vec{y}$ pueden ser continuos donde cada $y_i \in \mathbb{R}$ en los problemas de regresión o categóricos donde cada $y_i \in \mathbb{N}$ en los problemas de clasificación (en este caso y_i será una clase).

Definimos nuestro conjunto de datasets totales definido como $\mathbb{D}_{totales} = \{D_1, D_2, \ldots, D_n\}$ donde n es el número de datasets. En nuestro caso, se han utilizado 27 datasets de clasificación binaria para la realización de los experimentos de la sección 6.

4.1.2. Clasificadores Candidatos

Se utilizó el conjunto de datasets totales $\mathbb{D}_{totales}$ para evaluar un conjunto de clasificadores candidatos $\mathbb{C} = \{C_1, C_2, \ldots, C_n\}$ donde C_i es un clasificador.

Un clasificador es un algoritmo de *Machine Learning* que es capaz de estimar cual es la clase y_i de una instancia i de un dataset D a partir del vector de características $\vec{x_i}$ de esa instancia y del vector de hiperparámetros $\vec{\lambda}$ del clasificador (Dicho de otro modo, $C(\vec{x}, \vec{\lambda}) = y$). Agunos clasificadores necesitan ser entrenados con un conjunto de datos de entrenamiento definido como $D_{entrenamiento}$ para poder realizar dichas predicciones. Este dataset será un subconjunto de las instancias de un dataset D para utilizar el resto de ellas para su evaluación (definiremos este conjunto de datos como $D_{validacion}$)

Los clasificadores candidatos que se han utilizado en el entrenamiento son SVM, el perceptron multicapa (*Multilayer Perceptron* (MLP)) y el árbol de decisión. A continuación se explicará el funcionamiento de cada uno de ellos.

SVM

Una SVM es un modelo de ML utilizado tanto en problemas de clasificación como en problemas de regresión. En nuestro caso, definiremos una SVM lineal como un clasificador C_{svm} que utilizará un vector de características $\vec{x}$ para predecir una clase y que puede tomar valores $y = 1$ e $y = -1$. Para ello, se separará el espacio dimensional del conjunto de entrenamiento del dataset D mediante hiperplanos de margen máximo definidos como un conjunto de puntos que satisfacen $\vec{x}\ \vec{w} \cdot \vec{x} - b = 0$ donde $\vec{w}$ es el vector normal del hiperplano y $\vec{x}$ es un vector de soporte y $\frac{b}{\|\vec{w}\|}$ es el margen de los puntos en $\vec{x}$.

Cuando los datos del entrenamiento son linealmente separables, obtendremos el hiperplano minimizando $\|\vec{w}\|$ para que se cumpla

$$y_i(\vec{w} \cdot \vec{x} - b) \geq 1$$

para $i = 1, \ldots, n$.

En otro caso obtendremos el hiperplano minimizando $\|\vec{w}\|$ para que se cumpla:

$$\frac{1}{2}\|\vec{w}\|^2 + \lambda\left[\frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i(\vec{w}\cdot\vec{x} - b))\right]$$

MLP

Un perceptron multicapa es un modelo de ML utilizado tanto en problemas de clasificación como en problemas de regresión. En nuestro caso, definiremos el perceptron multicapa como un clasificador C_{mlp} que utilizará un vector de características $\vec{x}$ para predecir una clase y mediante una función de activación ReLU $y(z_n) = max(0, z_n)$ siendo z_n la salida de la última neurona de la última capa n. Cada neurona z_{ij} donde i representa la capa y j la neurona, se calcula su salida como:

$$y(z_{ij}) = max(0, b_{ij} + \sum_{j=0}^{n} y(z_{i-1j})w_{i-1j})$$

siendo w_{i-1j} un peso de MLP y b_{ij} el bias de la capa i y de la neurona j. Se entrenará el perceptron multicapa mediante el descenso del gradiente modificando los pesos tal que

$$\triangle w_{ji}(n) = -\eta\frac{\partial\mathcal{E}(n)}{\partial z_j(n)}y_i(n)$$

donde η es el ratio de aprendizaje y $\mathcal{E}(n)$ es la función de coste.

Árbol de Decisión

Un arbol de decisión es un modelo de ML utilizado tanto en problemas de clasificación como en problemas de regresión. En nuestro caso, definiremos el arbol de decisión como un clasificador C_{dt} que utilizará el vector de características $\vec{x}$ para predecir una clase y mediante divisiones de un dataset de entrenamiento D. Para ello, se elegirá la característica x_i que realizará la separación en función de una métrica $m = \{entropia, gini\}$.

4.1.3. Evaluación

Para la evaluación de cada uno de los clasificadores candidatos $\mathbb{C}$ en un subconjunto de datos de validación $D_{validacion}$ se necesitará una métrica que mida el porcentaje de aciertos, la precisión. La precisión se puede definir como

$$acc = \left(1 - \sum_{i=1}^{n}\frac{|y - y\prime|}{n}\right)$$

donde y es la clase del dataset D, $y\prime$ es la estimación del clasificador C obtenida mediante $C(\vec{x}, \vec{\lambda})$

y n es $|D_{validacion}|$.

La evaluación de nuestros clasificadores candidatos $\mathbb{C}$ se realizó mediante validación cruzada. En un dataset D, se divide aleatoriamente el dataset en K particiones para obtener un conjunto de particiones de entrenamiento $\{D^{(1)}_{entrenamiento}, \ldots, D^{(K)}_{entrenamiento}\}$ y un conjunto de particiones de validación $\{D^{(1)}_{validacion}, \ldots, D^{(K)}_{validacion}\}$ tal que $D^{(i)}_{entrenamiento} = D \backslash D^{(i)}_{validacion}$. De esta manera, la evaluación de la validación cruzada se define como

$$eval = \frac{1}{K} \sum_{i=1}^{K} acc(C, D^{(i)}_{validacion})$$

donde C es el clasificador evaluado.

4.1.4. *Feature Extraction*

Feature Extraction es una fase de ML que consiste en crear nuevos atributos a partir de los atributos originales de un dataset D. En nuestro caso, por cada uno de los datasets del conjunto $\mathbb{D}_{totales}$ se han extraido sus metacaracterísticas mediante las estadísticas de las distribuciones, que se han obtenido a partir de cada uno de los extractores de características, frente a los atributos del dataset (este conjunto se definirá como $\mathbb{MF}_{estadisticas}$). Además, también se han incluido al conjunto de metacaracterísticas totales la entropía del vector de clases $\vec{y}$, el número de instancias y el número de atributos de un dataset D definendo así este conjunto de metacaracterísticas como $\mathbb{MF}_{dataset} = \{instancias(D), atributos(D), entropia(\vec{y})\}$. Por tanto, el conjunto de metacaracterísticas totales se define como:

$$\mathbb{MF} = \mathbb{MF}_{estadisticas} \cup \mathbb{MF}_{dataset}$$

Para la obtención de $\mathbb{MF}_{estadisticas}$, definimos el conjunto de los extractores de características como $\mathbb{F} = \{f_1, f_2, \ldots, f_n\}$, siendo f un estadístico que operará sobre la distribución de un vector de atributos $\vec{a}$. En nuestro caso definiremos estos estimadores como $\mathbb{F} = \{\mu, \sigma, asimetria, curtosis\}$ siendo μ y σ, la media y la desviación típica. Por cada atributo $a \in D$ se aplicará un estadístico $f_i \in \mathbb{F}$ para obtener una distribución de este estimador definida como $E_i = \{f_i(\vec{a_1}), f_i(\vec{a_2}), \ldots, f_i(\vec{a_n})\}$. Del mismo modo, $\mathbb{MF}_{estadisticas}$ se obtendrán aplicando un vector de estadísticos definido como $\vec{s} = \{\mu, \sigma, min, Q1, Q2, Q3, max, asimetria, curtosis\}$ por cada distribución del los estimadores en $\mathbb{F}$ tal que:

$$\mathbb{MF}_{estadisticas} = \{s_1(E_i), \ldots s_n(E_i), s_1(E_{i+1}), \ldots, s_n(E_{i+1}), \ldots, s_1(E_m), \ldots s_n(E_m)\}$$

4.1.5. Metadataset

Tras aplicar *Feature Engineering* y evaluar los clasificadores candidatos $\mathbb{C}$ en un dataset $D_i \in \mathbb{D}_{totales}$, obtenemos un vector de metacaracterísticas $\vec{mf}_i$ y un vector de precisiones de los clasificadores candidatos $\vec{acc}_i$. Si aplicamos estos pasos en el conjunto de datasets totales $\mathbb{D}_{totales}$ para todo $i = 1, \ldots, n$ donde $n = |\mathbb{D}_{totales}|$, obtenemos el metadataset del combinador de clasificadores definido como:

$$M_{combinador} = \{(\vec{mf}_1, \vec{acc}_1), \ldots, (\vec{mf}_n, \vec{acc}_n)\}$$

Para obtener el metadataset del clasificador de clasificadores, tenemos que obtener un vector de clases $\vec{y}_d = \{y_{d1}, \ldots, y_{dn}\}$ en el cada clase y_{di} (donde i es una instancia del metadataset $M_{combinador}$) respresente el clasificador candidato que mejor precisión haya obtenido. Esto será posible aplicando $y_{di} = argmax(\vec{acc}_i)$ para todo $i = 1, \ldots, n$ siendo n el numero total de instancias del metadataset $M_{combinador}$. De esta forma, definiremos el metadataset del clasificador de clasificadores de tamaño n como:

$$M = \{(\vec{mf}_1, y_{d1}), \ldots, (\vec{mf}_n, y_{dn})\}$$

De la misma forma que hemos hecho con los datasets, también podemos definir este metadatasets como:

$$M = \{\vec{a}_{mf1}, \ldots, \vec{a}_{mfn}, \vec{y}_d\}$$

4.1.6. Normalización de los datos

Para realizar los experimentos 1.1 y 1.2 descritos en la sección 6, se han tenido que normalizar los datos del metadataset del clasificador de clasificadores M aplicando para todo $i = 1, \ldots, n$, donde n es el número de atributos de M,

$$\vec{a}_{mf_i}\prime = \frac{\vec{a}_{mf_i} - \mu}{\sigma}$$

donde μ y σ representan la media y la desviación típica de la distribución del vector de valores del atributo $\vec{a}_{mf_i}$. De esta manera conseguiremos que $\mu(\vec{a}_{mf_i}) = 0$ y $\sigma(\vec{a}_{mf_i}) = 1$ para todo $i = 1, \ldots, n$. Con esto definiremos el metadataset del clasificador de clasificadores con atributos normalizados como:

$$M\prime = \{\vec{a}_{mf1}\prime, \vec{a}_{mf2}\prime, \ldots, \vec{a}_{mfn}, \vec{y}_d\prime\}$$

4.1.7. *Feature Selection*

Para la obtención de las metacaracterísticas más importantes se han utilizado un conjunto de árboles de decisión $\mathbb{C}_dt = \{C_{dt1}, \ldots, \{C_{dtn}\}$ siendo n el numero de árboles de decisión totales para evaluar la importancia de cada una de las metacaracterísticas siguiendo [13].

4.1.8. Clasificador de Clasificadores

Se define nuestro clasificador de clasificadores como C_{cls} donde, a partir de un conjunto de clasificadores candidatos $\mathbb{C}$ y un dataset D, podemos predecir una clase y_{cls} que represente el mejor clasificador C siendo $C \in \mathbb{C}$ tal que:

$$C_{cls}(\vec{x}, \mathbb{C}) = y_{cls}$$

El clasificador de clasificadores C_{cls} será entrenado con el metadataset $M\prime$ en caso de que se quiera entrenar el clasificador con el metadataset normalizado y será entrenado con el metadataset M en caso de que se quiera entrenar el clasificador con el metadataset sin normalizar.

4.1.9. Combinador de Clasificadores

Se define nuestro combinador de clasificadores como C_{comb} donde a partir de un conjunto de clasificadores candidatos $\mathbb{C}$ y de un vector de regresores $\vec{R} = \{R_1, \ldots, R_n\}$ de tamaño $n = |\mathbb{C}|$ podemos predecir una clase y_{comb} tal que:

$$C_{comb}(\vec{x}, \mathbb{C}, \vec{R}) = y_{comb}$$

siendo $\vec{x}$ el vector de características de la instancia de y_{comb}. Para ello, nuestro combinador C_{comb} ha sido entrenado a partir del metadataset $M_{combinador}$ entrenando el regresor R_i por cada $i = 1, \ldots, n$ donde $n = |\mathbb{C}|$ el metadataset $M_{train} = \{\vec{a}_{mf1}, \ldots, \vec{a}_{mfn}, a\vec{c}c_{ci}\}$ siendo $M_{train} \subset M_{combinador}$.

Para realizar las predicciones de un vector de características $\vec{x}$ de un Dataset D, se obtendrá las metacaracterísticas $\vec{mf}$ a partir de los métodos vistos en *Feature Extraction* y *Feature Selection*. Nuestro combinador C_{comb} estimará utilizando el vector de regresores $\vec{R}$ un vector de precisión acc_p tal que:

$$acc_p = \{R_1(\vec{mf}), \ldots, R_n(\vec{mf})\}$$

Este vector de precisión acc_p se utilizará para calcular la política de importancia $\pi = \{w_1, \ldots, w_n\}$ donde cada w_i se calcula como

$$w_i = \frac{min(1, (10(max(\vec{acc}_p) - acc_{pi}) + 1)^{log((max(\vec{acc}_p) - acc_{pi}) + \epsilon)}}{\sum_{i=0}^{n} min(1, (10(max(\vec{acc}_p) - acc_{pi}) + 1)^{log((max(\vec{acc}_p) - acc_{pi}) + \epsilon)}}$$

siendo $\epsilon = 0{,}000001$. De esta forma podemos calcular y_{comb} a partir de un vector de características $\vec{x}$ utilizando el conjunto de clasificadores $\mathbb{C} = \{C_1, C_2, \ldots, C_n\}$ mediante:

$$y_{comb} = w_1 C_1(\vec{x}) + \cdots w_n C_n(\vec{x})$$

4.2. Diseño técnico

A continuación, en esta subsección vamos a exponer la comunicación de los procesos más importantes de nuestro modelo. Hemos dividido el diseño técnico en 4 procesos (A,B,C y D) y los representaremos más adelante por diagramas de flujo de datos que irán mostrando su complejidad a medida que avancemos.

Tanto el **Proceso B** como el **Proceso C** se han definido en la sección A del apéndice. Estos procesos que corresponden al entrenamiento del clasificador de clasificadores y del combinador de clasificadores, no son tan relevantes debido a que la mayoría de sus procesos internos son idénticos a los Procesos A y D.

El primer proceso que trataremos es el **Proceso A** que es el que engloba la predicción del clasificador de clasificadores y es uno de los objetivos principales (O1) definidos en la sección 3. Podemos ver el diagrama de flujo de datos de este proceso en la siguiente figura.

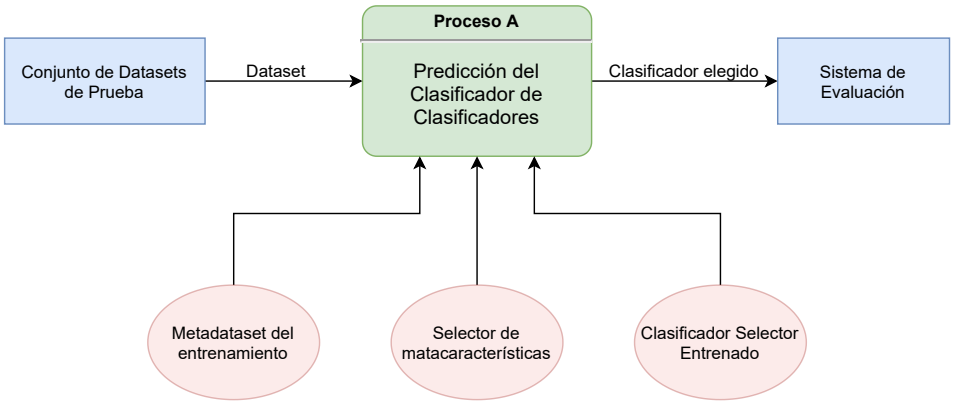

Figura 4.1: Diagrama de flujo de datos de Nivel 0 de la predicción del Clasificador de Clasificadores

La entrada de este proceso es un dataset obtenido por nuestro conjunto de datasets y su salida es una clase que representa el clasificador que estima el clasificador de clasificadores que conseguirá mejores resultados. Para este proceso se necesitará que el clasificador selector esté entrenado, un selector de metacaracterísticas y el metadataset que se ha utilizado en el entrenamiento.

Para entrar en detalle en el **Proceso A**, expondremos a continuación el diagrama de flujo de datos

de nivel 1 de este proceso.

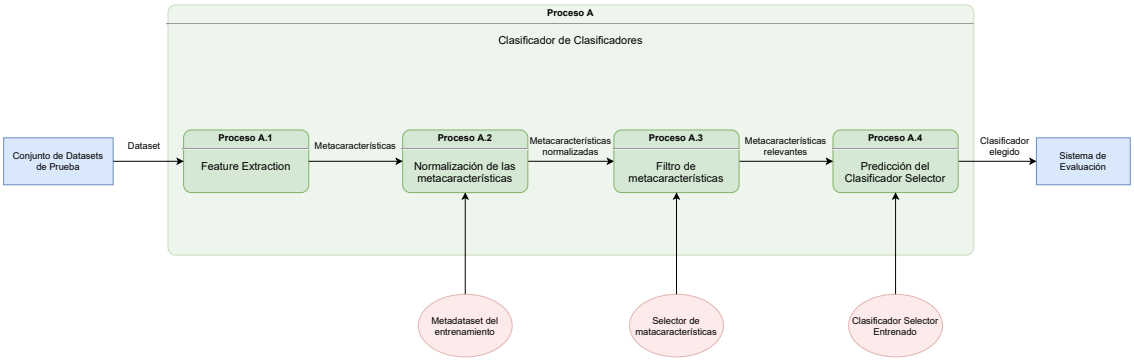

Figura 4.2: Diagrama de flujo de datos de Nivel 1 de la predicción del Clasificador de Clasificadores

Como podemos observar, el proceso A está compuesto por 4 procesos: **el Proceso A.1**, **el Proceso A.2**, **el Proceso A.3** y **el Proceso A.4**.

En el proceso A, el dataset del conjunto de datasets de prueba se envía al **Proceso A.1** donde se encargará de obtener las metacaracterísticas de este. A continuación, el **Proceso A.2** normaliza las metacaracterísticas gracias al metadataset original que se utilizó en el entrenamiento. Para ello se añadirá como una instancia extra al metadataset y se realizará la normalización de acuerdo a como se ha descrito en la subsección anterior. El **Proceso A.3** recoge esas metacaracterísticas y selecciona las más importantes con la ayuda de un selector de metacaracterísticas (obtenido tras ser entrenado en el proceso B). Por último, el clasificador selector estima en el **Proceso A.4** cual es el mejor clasificador a partir de las metacaracterísticas recibidas tras el proceso anterior. Más adelante, el clasificador elegido servirá para la evaluar si el clasificador de clasificadores acertó con su predicción.

El funcionamiento del **Proceso A.1** (que será el mismo para el Proceso D.1) se puede exponer con más profundidad en el siguiente diagrama de flujo de datos de nivel 2.

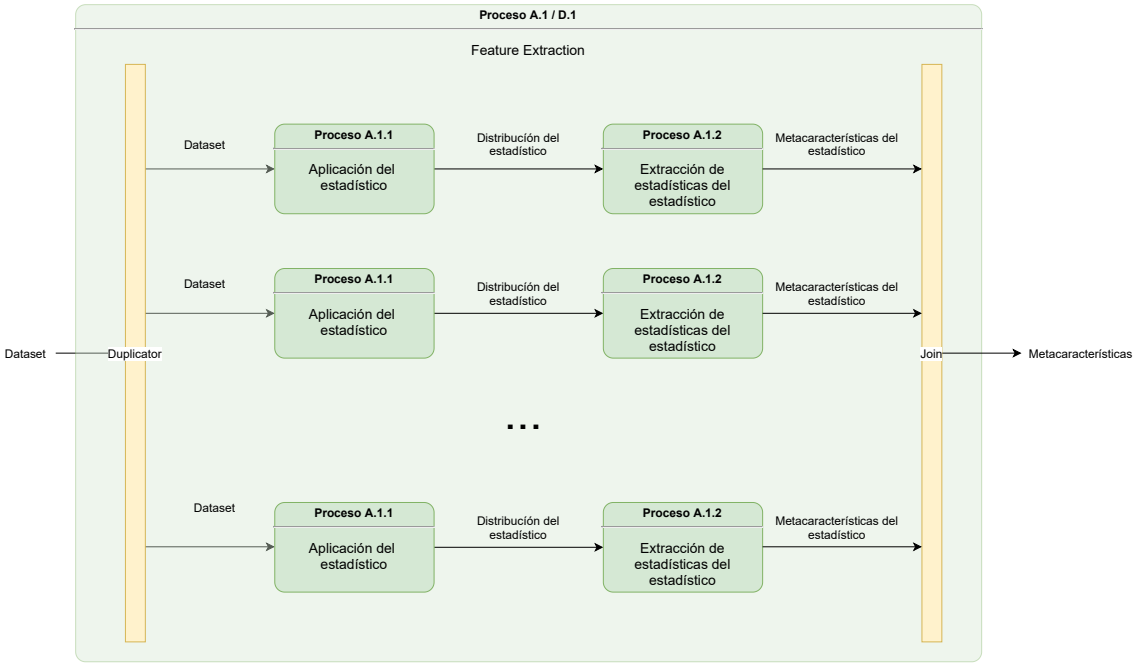

Figura 4.3: Diagrama de flujo de datos de Nivel 2 del Proceso A.1 y el Proceso D.1

El proceso **Proceso A.1** está compuesto por 2 procesos internos: **Proceso A.1.1** y **Proceso A.1.2**.

Para obtener las metacaracterísticas a partir de un dataset, se duplicará este conjunto de datos tantas veces como estimadores (definidos como extractores de características en la subsección 4.1) hayan. Este dataset servirá de entrada para la aplicación de un extractor de características sobre los atributos de dicho dataset (**Proceso A.1.1**). La salida de este proceso, que es una distribución de las estadísticas obtenidas por ese extractor de estadísticas sobre cada una de las distribuciones de los atributos, se utilizará como entrada para el **Proceso A.1.2**. Este proceso se encargará de obtener las metacaracterísticas del dataset original mediante la aplicación de un segundo grupo de estadísticos a la entrada del proceso. Para finalizar, se recogen las metacaracterísticas obtenidas por todos los extractores de características.

Por otro lado está **Proceso D**, que es el responsable de la predicción del combinador de clasificadores. Podemos observar los componentes principales de este proceso en la siguiente figura.

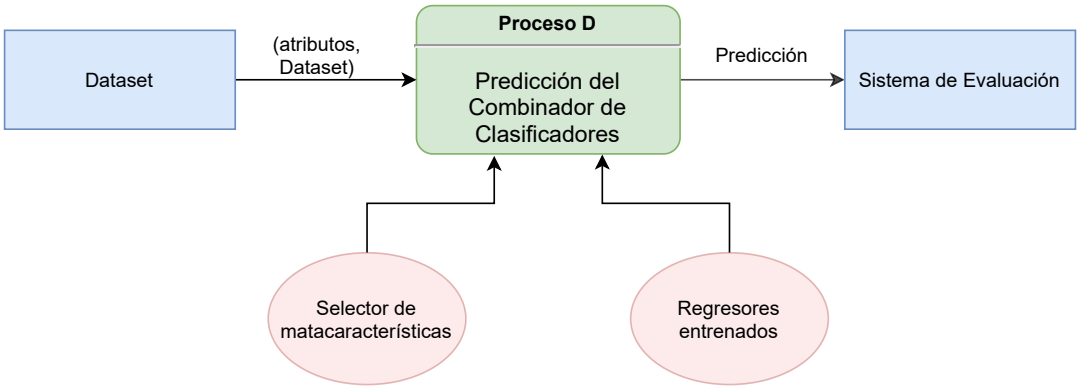

Figura 4.4: Diagrama de flujo de datos de Nivel 0 de la predicción del Combinador de Clasificadores

Normalmente, en la predicción de una instancia de un clasificador o un regresor solo se necesitan los atributos de esta. Sin embargo, el **Proceso D** requerirá del dataset de esos atributos por lo que definiremos la entrada de este proceso como una dupla que contiene los atributos de la instancia y el dataset al que pertenece. La salida de este proceso será predicción del combinador de clasificadores que se utilizará más adelante para la evaluación de nuestro clasificador. Además también se necesitará un selector de metacaracerísticas y que los regresores del combinador de clasificadores estén entrenados.

A continuación, representaremos el funcionamiento del **Proceso D** mediante el siguiente diagrama de flujo de datos de nivel 1.

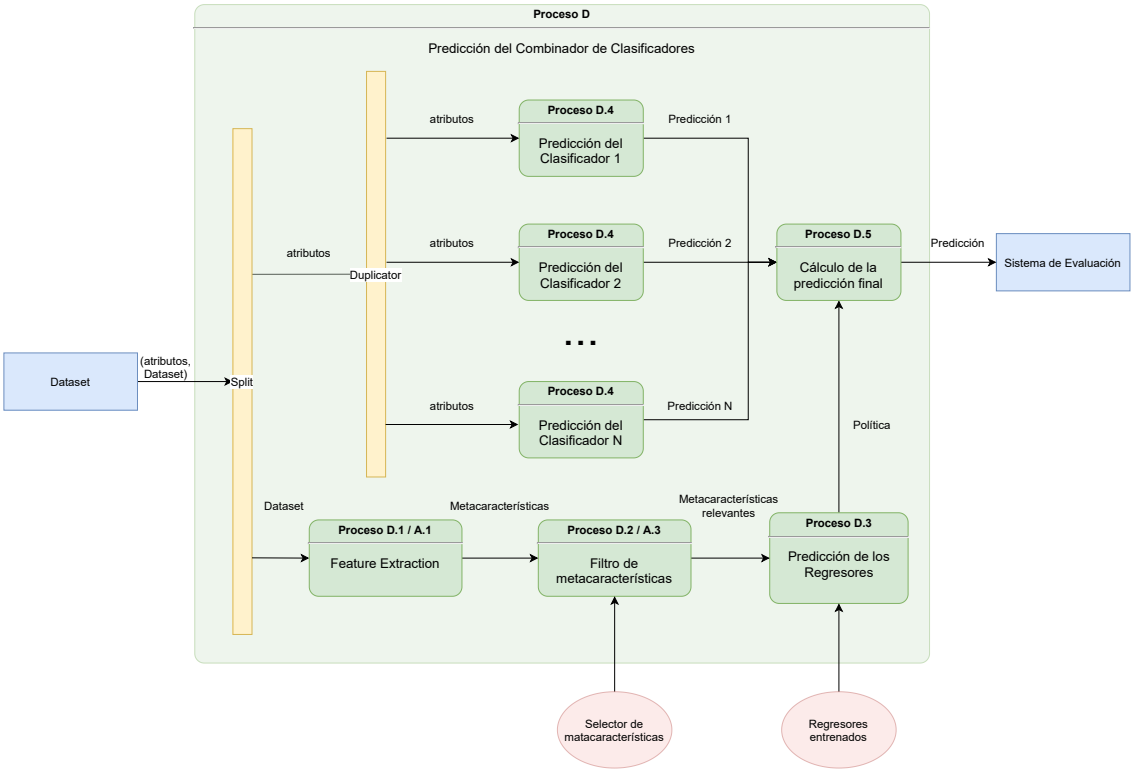

Figura 4.5: Diagrama de flujo de datos de Nivel 1 de la predicción del Combinador de Clasificadores

El **Proceso D** está compuesto por 5 procesos internos los cuales son: **Proceso D.1**, **Proceso D.2**, **Proceso D.3**, **Proceso D.4** y el **Proceso D.5**.

Al principio del **Proceso D**, se separa la dupla de la entrada del proceso para tratar estos datos de forma diferente.

Por un lado, el dataset servirá de entrada al **Proceso D.1** para extraer sus metacaracterísticas. Estas metacaracterísticas se utilizarán junto a un selector de metacaracterísticas para seleccionar las más importantes (**Proceso D.2**) que servirán como entrada para el **Proceso D.3**. Este proceso guardará las regresiones obtenidas por los regresores entrenados en las metacaracterísticas en una política de importancias tal como se define en la subsección anterior. Esta política se utilizará como una de las entradas del **Proceso D.5**

Por otro lado, los atributos se han multiplicaran tantas veces como el número de clasificadores del candidatos del combinador de clasificadores. Por cada uno de estos clasificadores se realizan predicciones a partir de los atributos mediante el **Proceso D.4** y se almacenan todas esas predicciones para que sirvan de entrada al **Proceso D.5**. En este proceso se calcula la predicción total ponderando cada una de las predicciones anteriores utilizando los pesos de la política de importancias.

4.3. Planificación del proyecto

Una vez visto tanto el diseño funcional como el diseño técnico, en esta subsección expondremos como se ha desarrollado nuestro proyecto e indagaremos en las fases por las que ha pasado. Para empezar, se mostrará un plano general con la duración de las tareas y más adelante, se explicará un poco más en detalle en que ha consistido cada una de ellas.

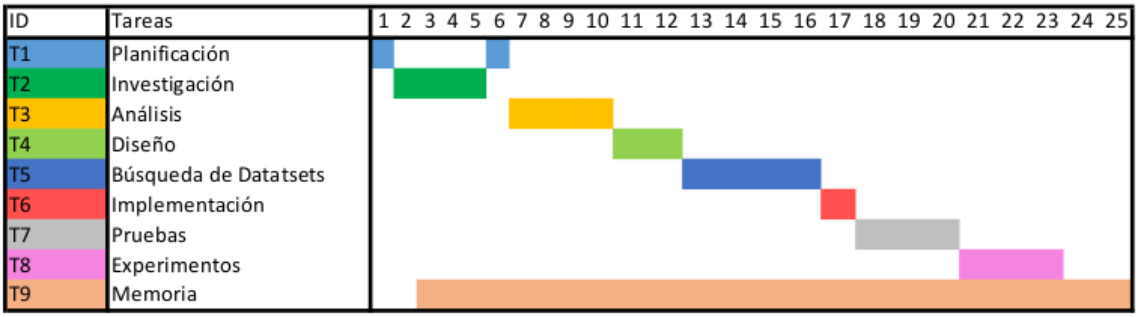

Figura 4.6: Diagrama de Gantt con las diferentes fases del proyecto. En el eje de abcisas se representa el tiempo de ocupación de una tarea en bloques de 12 horas.

Por un lado, este proyecto ha tenido una duración aproximada de 300 horas y desde el principio, se estimaron los pasos para su realización así como su tiempo. Como se puede ver en el diargama de Gantt (Figura 4.6), se han dividido las 300 horas en 25 bloques de 12 horas para abreviar y se ha propuesto una metodología en cascada para la realización de las tareas. No obstante, se ha contado con cierta flexibilidad durante el desarrollo del proyecto por lo que, aunque no se pueda apreciar en el diagrama, también se ha contado con cierta retroalimentación en cada uno de los pasos.

Lo primero que se ha hecho al comienzo del trabajo ha sido la fase de T1-Planificación. Este paso ha consistido en definir los objetivos e hipótesis iniciales así como definir la idea del proyecto. Más adelante, cuando finalizó la tarea T2, se volvió a replanificar las especificaciones del proyecto una vez estudiado los diferentes métodos para lograr nuestros objetivos. En esta segunda ocasión, se definieron las asunciones de la sección 3 para limitar el alcance del proyecto.

La segunda tarea que se realizó fué la T2-Investigación. En este paso se ha reunido toda la información relacionada con la sección 2 acerca de los diferentes proyectos del campo de AutoML. En esta fase en concreto se estudió diferentes formas de obtener las metacaracterísticas, optimizar los hiperparámetros (que al final no se aplicó directamente en la sección 6), selección de los atributos más relevantes, algoritmos de generación automática del *Pipeline* (tampoco tuvo lugar en el desarrollo) entre otros métodos.

Una vez realizada la investigación profunda en este area. Se realizó una captura de requisitos necesarios para satisfacer los objetivos determinados en la T1-Planificación (T3-Análisis). Por un lado, los requisitos funcionales abarcaron los procesos principales planteados en la subsección anterior

mientras que los requisitos no funcionales se centraron en el tiempo de ejecución de los algoritmos para evitar grandes demoras en las fases de T7-Pruebas y en T8-Experimentos. Sin embargo, debido a la ausencia de interacciones de usuario, no se realizó un estudio de los casos de uso.

Con los requisitos establecidos, se puso en marcha la fase de T4-Diseño, donde se especificaron los diferentes archivos que contendría el proyecto además de los diferentes diagramas realizados en la subsección de Diseño Técnico (4.2).

Una vez se plasmó físicamente el diseño de nuestro modelo, el siguiente paso era obtener los mejores datasets para llevar a cabo los experimentos. Esta fase en concreto se explicará en la posterior subsección 5.3 por lo que no entraremos tanto en detalle en explicarlo aquí.

Sin embargo, antes de coleccionar el mayor número de datasets posible, se pasó primero a la tarea T6-Implementación con el fin de implementar un evaluador para poder elegirlos y, una vez finalizada la tarea T5-Búsqueda de Datasets, se implementó el resto de las funcionalidades que quedaban. Como ya habíamos especificado el diseño del proyecto en T4, se redució la demora de tiempo y la dificultad de este paso.

Tras la implementación de todos los elementos necesarios para lograr nuestro propósito, se desarrollaron un conjunto de pruebas para encontrar los posibles errores que se hubieran cometido durante la fase anterior así como comprobar la correcta funcionalidad especificada en la fase de T3-Análisis. Cabe destacar que estas pruebas se han centrado en su mayoría en cada parte del proyecto por separado.

Por último, se realizó una serie de experimentos con el fin de corroborar las hipótesis planteadas y estudiar las posibles direcciones que podrían continuar con esta investigación.

5

Implementación

Una vez visto el diseño, presentamos toda la información referente a la implementación dividida en tres subsecciones. Primero, expondremos toda la información acerca del entorno en la subsección 5.1. En la subsección 5.2, listaremos los archivos que han contribuido en nuestra investigación y una breve explicación de su funcionamiento. Por último, explicaremos la importancia y la obtención de los conjuntos de datos que se han utilizado en la subsección 5.3.

5.1. Entorno

En esta subsección, se mostrarán las características del sistema operativo utilizado y después se presentará el lenguaje de programación de nuestro código junto con los paquetes externos importados.

El sistema operativo que se ha utilizado en todo momento ha sido *Windows*. Las especificaciones del sistema operativo son:

- Edición: Windows 10 Pro
- Versión: 1909
- Compilación del sistema operativo: 18363.900

El lenguaje de programación utilizado para el desarrollo del código ha sido Python (versión 3.6.5) en su totalidad. La elección de este lenguaje frente a los demás se ha basado en las facilidades que nos ofrecen paquetes externos como *scikit-learn* para integrar algunos algoritmos de *Machine Learning*. Además, también se ha tenido en cuenta que facilitaría la implementación de algunos módulos ya que Python admite herencia de clases.

Los paquetes que han importado durante la implementación son:

- scikit-learn (0.22.1): Para los clasificadores, la normalización del metadataset y la selección de las metacaracterísticas más relevantes.
- numpy (1.18.0): Para el manejo de los datasets y el cálculo de estadísticos durante la obtención de las metacaracterísticas.
- jupyter-client (6.0.0): Para la elaboración de los experimentos.

- jupyter-core (4.6.3)
- matplotlib (3.2.1): Para las gráficas de los experimentos
- scipy (1.4.1): Para el cálculo de algunos estadísticos durante la obtención de las metacaracterísticas.

5.2. Código

A continuación vamos a listar los archivos que se han utilizado durante el desarrollo de este proyecto y lo enlazaremos con algunos diagramas de flujo de la subsección 4.2. La implementación se ha basado en un diseño escalable para en futuras modificaciones realizar los mínimos cambios posibles. También cabe añadir que sólo se mencionarán los archivos y las principales características de estos que al final se han usado en los experimentos de la sección 6. Antes de empezar, se proporcionará un breve resumen de la funcionalidad de estos archivos.

Dentro de nuestro proyecto podemos dividir los archivos utilizados en:

- Datasets
- Módulos
- Experimentos
- Scripts

Los datasets estan almacenados con un formato especial debido a que en las primeras fases de prueba se hicieron experimentos con algunos clasificadores propios que necesitaban saber si los atributos eran continuos o categóricos. Estos archivos se encuentran en el directorio $\backslash data$ y son los encargados de entrenar los clasificadores y el clasificador de clasificadores.

Los módulos encapsulan la funcionalidad de nuestro proyecto así como la definición de algunas clases que nos sirvieron de utilidad. Estos se usarán para la realización de los experimentos los cuales estarán implementados en *Jupyter Notebooks* por comodidad.

Los scripts son los archivos que se han utilizado para facilitar algunas tareas como la generación de datasets artificiales (que al final no han tenido cabida en este proyecto) o la conversión de datos a tablas de LaTeX.

En esta subsección nos centraremos en explicar únicamente los archivos referentes a los módulos y a los experimentos los cuales son:

- AutoML.py
- Classifier.py
- Config.py
- Dataset.py
- Metadataset.py
- Metafeatures.py

- CombinedTest.ipynb
- DatasetTest.ipynb
- MetafeatureTest.ipynb
- ModelTests.ipynb

En **Dataset.py** se ha definido la clase *Dataset* para manejar la información de todos nuestros datos. Además, también hemos definido una clase *Fold* para almacenar los índices de cada partición que se usarán en la validación cruzada.

En **Classifier.py** se han implementado algunos clasificadores propios y se han añadido algunos clasificadores importados del paquete scikit-learn. Todos estos clasificadores heredarán de la clase *Classifier* donde se han definido los métodos necesarios para calcular su precisión y evaluar el modelo en función de los índices creados en *Dataset* (**procesos A, B, C y D**).

En **AutoML.py** se encuentran las implementaciones del clasificador de clasificadores, el combinador de clasificadores y el combinador de clasificadores uniforme que serán utilizados para los **procesos C y D**. Todos estos algoritmos heredarán de la clase *Classifier*.

En **Config.py** se han incluido todos los parámetros que han influido en la evaluación de los experimentos para facilitar su realización.

En **Metadataset.py** se ha definido la clase *Metadataset* que hereda de la clase Dataset. En esta clase extraerán los datasets del directorio $\backslash data$ y utilizará los módulos Metafeatures.py y Dataset.py para la creación y normalización (**procesos A.2 y B.3**) del Metadataset. Además, en este archivo se implementará el método para la selección de metacaracterísticas relevantes (**procesos A.3, B.4 y D.2**).

En **Metafeatures.py** se han definido todas las metacaracterísticas utilizadas junto a un método que las generará a partir de un Dataset (**procesos A.1 y D.1**).

Por último, los cuadernos **DatasetTest.ipynb** y **MetafeatureTest.ipynb** corresponden a los experimentos realizados en la subsección 6.1 y, del mismo modo, el experimento de la subsección 6.2 se ha realizado en **CombinedTest.ipynb**. Por comodidad, se han realizado una copia de los experimentos en un único cuaderno y se ha guardado en **ModelTest.ipynb**.

5.3. Conjunto de Datos

En esta subsección se expondrán los conjuntos de datos utilizados para los experimentos, así como su importancia, su obtención y su selección para llevar a cabo nuestros objetivos.

Los datos de entrada de cualquier proyecto o investigación de ML es el elemento más importante de todo el *pipeline* debido a que todo el proceso depende de esta parte fundamental. Por lo tanto, es necesario destacar que no servirá de nada mejorar el resto de tareas sin antes reparar en ello. A partir

de ahora nos referiremos a estos datos como *Datasets*.

En nuestro proyecto en concreto, elegiremos un subconjunto de los mejores datasets para realizar el entrenamiento del clasificador de clasificadores y el resto de datos se utilizarán para medir el rendimiento de nuestro modelo. Para ello, por cada uno de estos datasets obtendremos un conjunto de metacaracterísticas y lo asociaremos con una clase que representará el clasificador con mayor precisión en sus predicciones.

Los datasets se han obtenido en su mayor parte de OpenML aunque también se han probado medios y otros métodos como generación artificial de datos para su obtención. Tal como se señaló en la sección 3, se extrayeron únicamente los datasets que cumplieran con una serie de condiciones para reducir el espacio de búsqueda y enfocarnos en cumplir los objetivos que abarca nuestro problema. A continuación se expondrá una la lista con las condiciones más importantes:

Se considerarán solo los datasets de problemas:

- De clasificación binaria.
- Con un número de atributos totales entre 2 y 50.
- Con un número de instancias totales entre 400 y 10000.
- Con 0 *missing values*

Una vez se hayan extraido los datasets, para cada uno de ellos se realizará una evaluación de los clasificadores candidatos para la selección de modelos del clasificador de clasificadores. En el apéndice B se mostrará una tabla con los 50 datasets extraidos y la evaluación de cada clasificador tras realizar una validación cruzada de 10 particiones.

Una vez obtenidos los resultados, se realizará una segunda selección para obtener los mejores datasets para el entrenamiento. Se han tenido en cuenta una serie de reglas para esta selección y serán expuestas a continuación

- Al final, la selección de datasets resultante debe tener al menos un equilibro entre el número de clasificadores especializados para este conjunto. De no ser así, esto podría afectar en la calidad de los resultados en los experimentos.
- Los datasets no se parecerán entre ellos, por lo tanto no se eligirán 2 que vengan del mismo problema. Esto se hace para prevenir que hayan dos datasets con metacaracterísticas similares.
- Se eligirán los datasets con la mayor diferencia de precisión entre el primer y segundo mejor clasificador. Nos interesa elegir los datasets en los que se remarque la mayor diferencia entre usar un clasificador respecto a otro.

Para concluir con esta subsección cabe resaltar que la extracción de los datos utilizados ha sido un elaborado proceso de evaluación que nos ha demorado bastante más de lo que imaginábamos. No obstante, gracias al tiempo invertido no hemos asegurado en seleccionar los mejores datasets para nuestros experimentos, los cuales se expodrán más adelante en la sección de integración, pruebas y resultados (Sección 6).

6

Integración, pruebas y resultados

En esta sección expondremos las diferentes pruebas que se han realizado. Antes de comenzar con los experimentos, se presentarán los datasets que se han utilizado de acuerdo con las reglas de la subsección anterior (5.3) en la siguiente tabla.

Dataset	SVM	MLP	Árbol de Decisión
acutel.data	58.33	65.83	**100.00**
attr100.data	51.23	**79.38**	55.12
australian.data	73.77	76.52	**81.30**
autoUniv.data	**76.40**	73.20	66.90
bank-Marketing.data	**88.45**	87.33	86.80
breast-cancer.data	71.07	**74.33**	62.78
chscase-census6.data	**57.25**	56.00	51.50
chscase-funds.data	**68.19**	48.60	67.05
cloud.data	70.36	68.45	**90.00**
cmc.data	66.53	**71.28**	65.11
codebreaker.data	**76.60**	72.20	68.20
cpu.data	93.31	77.45	**97.62**
fri-c0-1000-5.data	90.60	**91.20**	82.50
german.data	**70.80**	68.20	69.60
ilpd.data	**71.34**	68.78	62.08
kin8nm.data	89.72	**91.02**	77.00
parity5-plus-5.data	29.00	**100.00**	71.54
piechart3.data	**87.56**	80.52	83.09
prnn-crabs.data	84.00	**98.00**	89.50
qsar-biodeg.data	82.19	**87.20**	82.09
rmftsa-sleepdata.data	67.77	55.27	**68.93**
space-ga.data	54.97	51.75	**77.12**
tae.data	65.67	66.96	**80.13**
unknown.data	81.42	73.11	**86.11**
vertebraC.data	**84.84**	81.29	80.97
vowel.data	96.77	**99.90**	98.08
wilt.data	94.61	97.48	**98.10**

Tabla 6.1: Precisión media de los clasificadores elegidos tras la validación cruzada con los datasets seleccionados para los experimentos

Como se puede observar, se han elegido 27 datasets que, al servir de evaluación para los clasificadores candidatos, el número de clasificadores que mejores resultados han obtenido frente a los demás es equilibrado (en este caso tenemos 9 mejores resultados por cada clasificador). A partir de ahora nos referiremos a estos datasets como datasets equilibrados.

Los clasificadores candidatos que se han utilizado al final han sido *Support Vector Machine*, el perceptron multicapa y el árbol de decisión aunque se podrían haber seleccionado otros. El propósito de los siguientes experimentos fueron corroborar las hipótesis planteadas en la sección 3.

Todos los experimentos realizados han seguido las especificaciones descritas en el diseño funcional (sección 4.1). En la selección de metacaracterísticas relevantes se ha utilizado 50 árboles de decisión para todos los experimentos.

6.1. Experimentos del Clasificador de Clasificadores

El objetivo de nuestro primer modelo es predecir que clasificador, de entre un conjunto de clasificadores candidatos, consigue los mejores resultados a partir de las metacaracterísticas obtenidas por un dataset. Por ese motivo, nuestros experimentos se centrarán en encontrar los mejores clasificadores para este propósito y se estudiará el número de datasets y metacaracterísticas adecuado para ello.

Además, se compararán los resultados obtenidos por las metacaracterísticas sin normalizar con las metacaracterísticas normalizadas para averiguar si es un factor que influye en el rendimiento de nuestro clasificador de clasificadores.

Para estos experimentos, los modelos que se probarán para elegir el mejor clasificador son los siguientes:

- Regresión Logística (solver: 'lbfgs')
- Support Vector Machine (SVM) (gamma: 'scale')
- KNN (k: 3)
- Árbol de Decisión
- Perceptron Multicapa (MLP) (capas: (64, 32, 1))
- Proceso Gaussiano (kernel: 7 RBF)
- Random Forest(estimadores: 160)

Los hiperparámetros de estos clasificadores fueron elegidos en base a pruebas no relevantes mediante *Guided Search*.

En cuanto a la elección del conjunto de datos de entrenamiento y el conjunto de datos de prueba cabe destacar que se ha utilizado una versión diferente de validación cruzada de la definida en la subsección 4.1. El método ha consistido en seleccionar aleatoriamente ,para el conjunto de validación, 6 dataset equilibrados (2 dataset por cada clasificador candidato) por cada una de las 100 particiones

que se utilizarán para ambos experimentos. En cuanto al conjunto de entrenamiento, tendrá un tamaño variable en el primer experimento y serán los 21 datasets restantes en el segundo.

En el caso de la selección de las metacaracterísticas más relevantes, se utilizarán diferentes conjuntos de metacaracterísticas en el segundo experimento y se limitará el uso a 15 metacaracterísticas en el primero para comparar los diferentes clasificadores de clasificadores.

6.1.1. Experimento 1.1: Mejor modelo en función del número del conjunto de datos

Descripción

El número de datasets que entrenará el clasificador de clasificadores es un valor muy importante a considerar ya que si el conjunto de entrenamiento no es lo suficientemente grande, nuestro clasificador de clasificadores no obtendrá buenos resultados. Por tanto, el propósito de este experimento es encontrar ese valor para el conjunto de clasificadores definido anteriormente y estimar si es necesario recopilar más datasets para mejorar el rendimiento de estos.

Para este experimento se han utilizado de 3 a 21 datasets en paquetes de 3 datasets equilibrados para el conjunto de entrenamiento. El resto de la configuración se ha definido anteriormente.

Resultados

A continuación presentaremos los resultados obtenidos del experimento en dos figuras. La primera, representará la evaluación de nuestro clasificador de clasificadores con las metacaracterísticas del metadataset sin normalizar y estará definida en la sección C del apéndice. Por otra parte, podemos observar el mismo experimento pero con la normalización de los datos del metadataset en la siguiente figura.

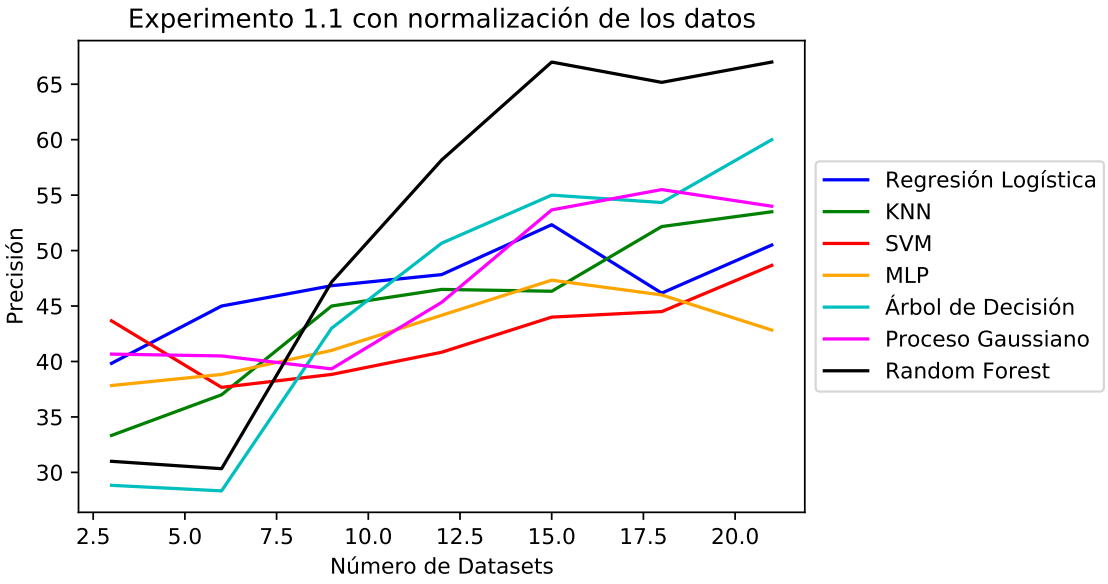

Figura 6.1: Experimento 1.1: Relación entre el número de datasets en el entrenamiento y precisión de los clasificadores con datos normalizados

Como se puede apreciar, normalizar los datos ha sido un factor relevante que ha influido en el rendimiento del clasificador de clasificadores.

Por un lado, algunos clasificadores han experimentado una mejora en su precisión a medida que se han utilizado más datasets en el entrenamiento como en el caso del árbol de decisión y SVM tras utilizar 21 datasets para este propósito (se han mejorado de 53.5 % a 60 % y de 32 % a 48,66 % respectivamente).

Además podemos observar que la forma del crecimiento de las evaluaciones es más regular tras aplicar la normalización en contraste con las evaluaciones obtentidas por los datos sin normalizar. En este caso, los clasificadores parecen que no progresar a la misma velocidad a excepción de *Random Forest* (que parece que evoluciona de la misma manera en ambos resultados) a medida que se aumenta el conjunto de entrenamiento. En contraste con este clasificador, la precisión de los procesos gaussianos va disminuyendo partir de los 12 datasets en los experimentos sin datos normalizados y es posible que siga el mismo comportamiento si se aumentara el número de datasets del entrenamiento en los experimentos con datos normalizados.

Los mejores clasifcadores que se han obtenido de este experimento son los realizados con datos normalizados tras 21 datasets y estos son *Random Forest* y el árbol de decisión con un 67 % y 60 % de precisión respectivamente. Los procesos gaussianos, que obtuvieron los mejores resultados en los experimentos de entre 3 a 9 datasets en el conjunto de entrenamiento, fueron los terceros usando tan solo 12 datasets con un 57.16 %.

A continuación se realizará en el siguiente experimento una comparación de las evaluaciones ob-

tenidas con diferentes cantidades de metacaracterísticas utilizando el máximo número de datasets posible tras las conclusiones obtenidas por estas pruebas.

6.1.2. Experimento 1.2: Mejor modelo en función del número de metacaracterísticas

Descripción

En este experimento se evaluarán los clasificadores de clasificadores utilizando diferentes números de metacaracterísticas (de 1 a 15) para estudiar los efectos que provocan en sus evaluaciones. El resto de la configuración del experimento se ha definido anteriormente al comienzo de la subsección.

Resultados

Como en el experimento anterior, mostraremos en gráficas los resultados de las evaluaciones de la validación cruzada del clasificador de clasificadores variando el número de metacaracterísticas durante el entrenamiento tanto con los datos del metadataset normalizados como sin ellos.Ambas gráficas han sido suavizadas haciendo una media con las precisiones adyacentes con el fin de reflejar mejor el comportamiento general de un clasificador cuando aumentan o disminuyen el número de datos utilizados. Definiremos los resultados sin datos normalizados en la sección C del apéndice como en el experimento anterior y, a continuación, definiremos los resultados del experimento realizado con datos normalizados en la siguiente figura.

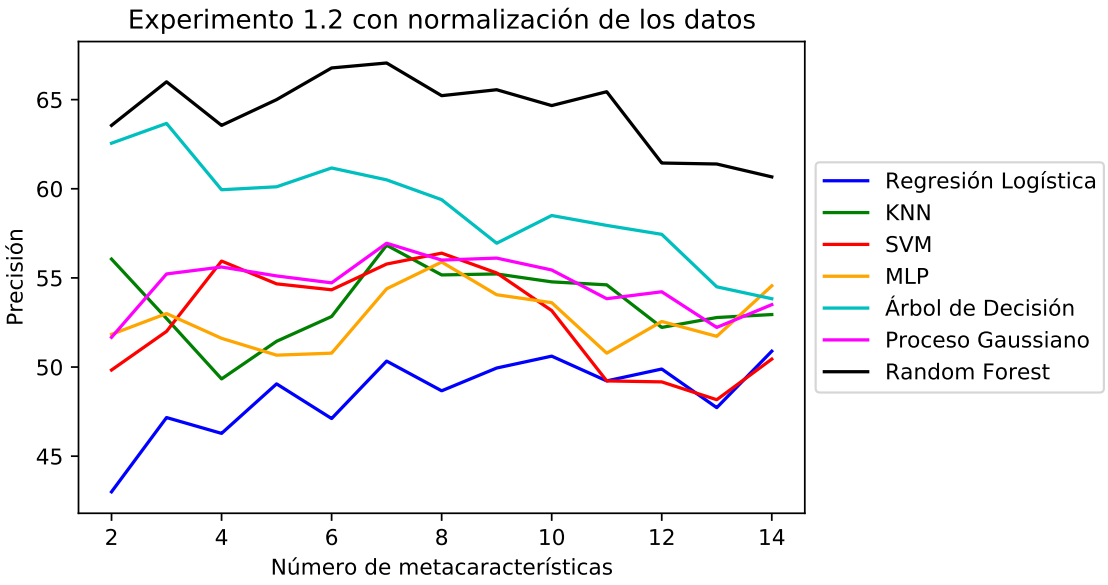

Figura 6.2: Experimento 1.2: Relación entre el número de metacaracterísticas relevantes en el entrenamiento y la precisión suavizada de los clasificadores de clasificadores

Tras la vista de los resultados obtenidos se podrían dividir la observaciones en dos temas.Por un lado, se pueden observar los efectos que provocan la normalización de los datos del metadataset en los experimentos. Por otro lado, se puede apreciar lo mismo con el número de metacaracterísticas.

Podemos recalcar que la normalización ha sido determinante en la evaluación de casi todos los clasificadores de clasificadores. Como se puede apreciar, uno de los clasificadores que más se ha beneficiado de la normalización es KNN tras reducir considerablemente la maldición de la dimensionalidad, aunque su mejor rendimiento haya sido 62 % utilizando únicamente una metacaracterística. Sin embargo, es notable que se han empeorado las precisiones del arbol de decisión y *Random Forest* tras usar un gran número de metacaracterísticas y aplicarse la normalización. A pesar de esto, la normalización ha permitido conseguir mejores resultados en casi todos los clasificadores a excepción de KNN y MLP. *Random Forest* ha conseguido los mejores resultados de nuevo con un 71.33 % seguido del árbol de decisión con un 70.05 %.

Por último, podemos ver que la mayoría de los mejores resultados de cada clasificador se han conseguido con muy pocas metacaracterísticas. Esto se puede ver en *Random Forest* tras conseguir su mejor precisión con tan solo dos de ellas.

6.2. Experimentos del Combinador de Clasificadores

En este caso, el propósito principal de estos experimentos es corroborar la hipótesis H2 cambiado el enfoque del problema. En lugar de utilizar un modelo para predecir que clasificador (entre un conjunto de clasificadores dado) va a obtener los mejores resultados para un determinado dataset, usaremos un modelo por cada clasificador candidato para predecir la precisión que obtendría de un dataset dado y así crear un clasificador que combine eficientemente estos clasificadores.

6.2.1. Experimento 2.1: Diferencias entre resultado combinado frente a resultados separados

Descripción

En este experimento, se compararán los rendimientos de los clasificadores presentados al principio de la sección con dos modelos que combinarán estos clasificadores para realizar sus predicciones. Estos modelos tendrán únicamente la diferencia de que el combinador de clasificadores uniforme utilizará todos los clasificadores candidatos por igual mientras que nuestro combinador no.

Para la evaluación de los combinadores presentados, se compararán los rendimientos de los clasificadores candidatos con estos. Para ello se evaluarán los clasificadores con cada uno de los datasets extraidos usando una validación cruzada de 10 particiones. Para las predicciones del combinador de

clasificadores en un datasets determinado se entrenarán los regresores con las metacaracterísticas de 24 datasets diferentes cuya evaluación de los clasificadores candidatos esté equilibrada

Resultados

A continuación se presentará parcialmente el resultado de nuestro experimento en la siguiente tabla. Por otra parte, expondremos la tabla del experimento completo en la sección C del apéndice.

Dataset	SVM	MLP	Árbol de Decisión	Comb. Uniforme	Nuestro Comb.
acutel.data	58.33	65.83	**100.00**	80.83 +- 24.17	91.67 +- 17.08
attr100.data	51.23	**79.38**	55.12	68.4 +- 5.86	67.42 +- 6.28
australian.data	73.77	76.52	81.30	81.01 +- 3.46	**82.03 +- 5.11**
autoUniv.data	76.40	73.20	66.90	77.1 +- 4.41	**77.4 +- 3.83**
bank-Marketing.data	**88.45**	87.33	86.80	88.79 +- 1.22	88.61 +- 1.23
breast-cancer.data	71.07	**74.33**	62.78	74.06 +- 5.37	74.07 +- 7.15
chscase-census6.data	**57.25**	56.00	51.50	53.75 +- 9.03	57.0 +- 7.97
chscase-funds.data	**68.19**	48.60	67.05	65.53 +- 9.02	67.16 +- 11.11

Tabla 6.2: Primeros 8 datasets evaluados por nuestros combinadores de clasificadores junto con la evaluación de los clasificadores candidatos.

Tras la realización de este experimento hemos conseguido en el 18.51 % de nuestros datasets mejores resultados con la combinación de los clasificadores que con el mejor clasificador candidato para ese Dataset. Además, aunque no se hayan conseguido los mejores resultados en el 81.49 % restante, podemos ver que se alcanza una mejor precisión que el segundo mejor clasificador candidato.

En comparación al combinador de clasificadores uniforme, nuestro combinador supera o iguala con un umbral de 0.1 % de precisión a este combinador en 18 de los 27 datasets extraidos gracias a la política de importancias de nuestro combinador. Sin embargo, podemos concluir que los resultados son aun así bastante parejos aunque el margen de mejora que obtenemos de nuestro combinador sea superior al combinador uniforme.

En la última sección se presentarán las conclusiones del proyecto y los posibles rumbos que puede tomar para futuros trabajos que continúen con esta línea de investigación.

7

Conclusiones y trabajo futuro

Para finalizar con este trabajo, en esta sección expondremos nuestra conclusión del proyecto. Después, contrastaremos los objetivos y las hipótesis establecidas con los resultados obtenidos y presentaremos las posibles direcciones que podrían tomarse en trabajos futuros.

Tal como se describió en la Introducción (Sección 1), nuestros modelos podrían ser de gran utilidad como seleccionador de clasificadores automático y podrían formar parte de cualquier *Pipeline* de *Machine Learning*. Esto podría acelerar considerablemente el desarrollo de estos proyectos puesto que no es necesario evaluar cada uno de los clasificadores para estimar cual sería el más adecuado.

A continuación repasaremos si se han logrado los objetivos definidos en la sección 3:

O-1.– Se implementará una herramienta que clasifique que clasificador usar para un nuevo conjunto de datos. Este objetivo se ha logrado con éxito.

O-2.– Se implementará un combinador de clasificadores que estimará el desempeño de cada clasificador candidato en función del dataset y lo combinará para las evaluaciones de las instancias. Este objetivo se ha logrado con éxito.

O-3.– Se recolectarán 18 datasets los cuáles, al evaluar cada clasificador candidato con estos, el número de clasificadores que mejores resultados han obtenido frente a los demás sea equilibrado. Este objetivo se ha logrado con éxito, de hecho hemos extraido al final 27 datasets de los 50 totales.

O-4.– Nuestro modelo seleccionará las metacaracterísticas más relevantes para predecir el clasificador y entrenar tanto el clasificador de clasificadores como el combinador de clasificadores. Este objetivo se ha logrado con éxito.

O-5.– Se comparará el rendimiento de diferentes clasificadores para cumplir el objetivo O1. Este objetivo se ha logrado con éxito.

O-6.– Se compararán los resultados obtenidos con algunos frameworks de AutoML. Este objetivo no se ha lograr debido a la restricción R1.

A la vista de los resultados obtenidos de las pruebas realizadas se expondrán las conlusiones a las que se han llegado y se contrastarán con las hipótesis establecidas.

- Tras la realización del experimento 1.1, parece que si se hubiera continuado aumentando los datasets en el conjunto de entrenamiento, habría una mejora en el redimiento de la mayoría de los clasificadores utilizados. Con esto hemos demostrado parcialmente que la hipótesis H4 se cumple para algunos clasificadores.
- Tras la realización del experimento 1.2, hemos comprobado que en la mayor parte de clasificadores utilizados

no necesitan muchas metacaracterísticas para obtener los mejores resultados. Con esto hemos corroborado parcialmente que la hipótesis H5 se cumple para algunos clasificadores.

- Tras la realización del experimento 1.2 y 1.1, hemos observado que normalizar los datos del metadataset permite mejorar los resultados en la mayoría de los clasificadores de clasificadores. Con esto hemos demostrado parcialmente que la hipótesis H3 se cumple para algunos clasificadores. Además, nos hemos percatado que al menos con los hiperparámetros utilizados, se han conseguido que algunos clasificadores superaran a los procesos gaussianos por lo que la hipótesis H6 no se ha podido corroborar en estas condiciones.
- Tras la realización del experimento 2.1, se ha demostrado que es posible combinar los clasificadores para obtener un clasificador que obtenga mejores resultados. Con esto hemos podido validar que la hipótesis H2 es verdadera.
- Tras la realización de todos los experimentos, se ha llegado a la conclusión de que es posible predecir que modelo se ajusta mejor a un problema a partir sus metacaracterísticas. Con esto hemos corroborado la hipótesis H1.

Con esto podemos concluir que se han alcanzado los objetivos propuestos a excepción de O6 por falta de tiempo. En cuanto a las hipótesis planteadas hemos podido corroborar 5 de ellas y aunque H6 no haya podido ser demostrada, es posible que con otros hiperparámetros se pueda llegar a confirmar.

Este trabajo puede servir de punto de partida para nuevos proyectos. A continuación expropondremos diferentes ampliaciones que se podrían hacer en nuestro trabajo dividiéndolas en ampliaciones de exploración y ampliaciones de explotación.

Por una parte, se podrían hacer ampliaciones de exploración cuyo propósito sería añadir nuevas técnicas y recursos para modificar nuestro diseño y así conseguir mejores resultados. Algunas de las ampliaciones que proponemos para este tipo son aumentar el conjunto de entrenamiento de nuestros modelos, conseguir los mejores hiperparámetros de los clasificadores candidatos mediante HPO y tratar el rendimiento de nuestros modelos como un problema CASH y así obtener los mejores parámetros de todos nuestros procesos con técnicas como la optimización bayesiana, algoritmos genéticos o PSO.

Por otra parte, las ampliaciones de explotación serían aquellas que buscarían mejorar nuestros modelos actuales sin necesidad de añadir nuevos recursos ni modificar el diseño. Para este tipo de ampliaciones, nosotros proponemos que se podría comparar los resultados con algunos *Frameworks* de ML (ya que fue el objetivo que no se pudo cumplir), probar diferentes clasificadores para el entrenamiento de nuestros modelos y comprobar si la normalización de datos mejoraría nuestro combinador de clasificadores.

Bibliografía

[1] R. L. Anderson, *Recent advances in finding best operating conditions*, Journal of the American Statistical Association, 48 (1953), pp. 789–798.

[2] J. Bergstra and Y. Bengio, *Random search for hyper-parameter optimization*, The Journal of Machine Learning Research, 13 (2012), pp. 281–305.

[3] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, *Algorithms for hyper-parameter optimization*, in Advances in neural information processing systems, 2011, pp. 2546–2554.

[4] L. Breiman, *Random forests*, Machine learning, 45 (2001), pp. 5–32.

[5] P.-W. Chen, J.-Y. Wang, and H.-M. Lee, *Model selection of svms using ga approach*, in 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541), vol. 3, IEEE, 2004, pp. 2035–2040.

[6] M. Claesen, J. Simm, D. Popovic, Y. Moreau, and B. De Moor, *Easy hyperparameter search using optunity*, arXiv preprint arXiv:1412.1114, (2014).

[7] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen, et al., *Evolutionary algorithms for solving multi-objective problems*, vol. 5, Springer, 2007.

[8] T. Dinsmore, *Automated machine learning: A short history, 2016*, URL https://blog. datarobot. com/automated-machine-learning-short-history.

[9] H. J. Escalante, M. Montes, and L. Villaseñor, *Particle swarm model selection for authorship verification*, in Iberoamerican Congress on Pattern Recognition, Springer, 2009, pp. 563–570.

[10] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, *Efficient and robust automated machine learning*, in Advances in neural information processing systems, 2015, pp. 2962–2970.

[11] E. C. Garrido-Merchán and D. Hernández-Lobato, *Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes*, Neurocomputing, 380 (2020), pp. 20–35.

[12] R. Gaudel and M. Sebag, *Feature selection as a one-player game*, 2010.

[13] P. Geurts, D. Ernst, and L. Wehenkel, *Extremely randomized trees*, Machine learning, 63 (2006), pp. 3–42.

[14] Y. Gil, K.-T. Yao, V. Ratnakar, D. Garijo, G. Ver Steeg, P. Szekely, R. Brekelmans, M. Kejriwal, F. Luo, and I.-H. Huang, *P4ml: A phased performance-based pipeline planner for automated machine learning*, in Proceedings of Machine Learning Research, ICML 2018 AutoML Workshop, 2018.

[15] M. Henao-Calad and V. Rodríguez-Lora, *Modelo de conocimiento conceptual como apoyo a la ingeniería del conocimiento*, Ingeniare. Revista chilena de ingeniería, 20 (2012), pp. 412–424.

[16] J. Y. Hesterman, L. Caucci, M. A. Kupinski, H. H. Barrett, and L. R. Furenlid, *Maximum-likelihood estimation with a contracting-grid search algorithm*, IEEE transactions on nuclear science, 57 (2010), pp. 1077–1084.

[17] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom, *Declarative support for sensor data cleaning*, in International Conference on Pervasive Computing, Springer, 2006, pp. 83–100.

[18] J. M. Kanter and K. Veeramachaneni, *Deep feature synthesis: Towards automating data science endeavors*, in 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), IEEE, 2015, pp. 1–10.

[19] G. Katz, E. C. R. Shin, and D. Song, *Explorekit: Automatic feature generation and selection*, in 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, 2016, pp. 979–984.

[20] A. Kaul, S. Maheshwary, and V. Pudi, *Autolearn—automated feature generation and selection*, in 2017 IEEE International Conference on data mining (ICDM), IEEE, 2017, pp. 217–226.

[21] B. Kégl, *How to build a data science pipeline, 2017*, URL https://www. kdnuggets. com/2017/07/build-data-science-pipeline. html.

[22] U. Khurana, D. Turaga, H. Samulowitz, and S. Parthasrathy, *Cognito: Automated feature engineering for supervised learning*, in 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), IEEE, 2016, pp. 1304–1307.

[23] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, *On the relationship between classical grid search and probabilistic roadmaps*, The International Journal of Robotics Research, 23 (2004), pp. 673–692.

[24] R. Lippmann, *An introduction to computing with neural nets*, IEEE Assp magazine, 4 (1987), pp. 4–22.

[25] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, *Progressive neural architecture search*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 19–34.

[26] H. Motoda and H. Liu, *Feature selection, extraction and construction*, Communication of IICM (Institute of Information and Computing Machinery, Taiwan) Vol, 5 (2002), p. 2.

[27] R. S. Olson and J. H. Moore, *Tpot: A tree-based pipeline optimization tool for automating machine learning*, in Automated Machine Learning, Springer, 2019, pp. 151–160.

[28] P. Pudil, J. Novovičová, and J. Kittler, *Floating search methods in feature selection*, Pattern recognition letters, 15 (1994), pp. 1119–1125.

[29] H. Putnam, *Models and reality*, The Journal of Symbolic Logic, 45 (1980), pp. 464–482.

[30] D. Pyle, *Data preparation for data mining*, morgan kaufmann, 1999.

[31] E. Rahm and H. H. Do, *Data cleaning: Problems and current approaches*, IEEE Data Eng. Bull., 23 (2000), pp. 3–13.

[32] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, *Regularized evolution for image classifier architecture search*, in Proceedings of the aaai conference on artificial intelligence, vol. 33, 2019, pp. 4780–4789.

[33] C. W. Reynolds, *Flocks, herds and schools: A distributed behavioral model*, in Proceedings of the 14th annual conference on Computer graphics and interactive techniques, 1987, pp. 25–34.

[34] B. Samanta, *Gear fault detection using artificial neural networks and support vector machines with*

genetic algorithms, Mechanical systems and signal processing, 18 (2004), pp. 625–644.

[35] R. Sannazzaro, *A distilled list of ai trends for 2020, 2019*, URL https://www. towardsdatascience. com/a-distilled-list-of-ai-trends-for-2020-e2fc83a9b092. html.

[36] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., *A general reinforcement learning algorithm that masters chess, shogi, and go through self-play*, Science, 362 (2018), pp. 1140–1144.

[37] P. Sondhi, *Feature construction methods: a survey. sifaka. cs. uiuc. edu 69*, (2009).

[38] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, *Auto-weka: Combined selection and hyperparameter optimization of classification algorithms*, in Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, 2013, pp. 847–855.

[39] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2, MIT press Cambridge, MA, 2006.

[40] B. Zoph and Q. V. Le, *Neural architecture search with reinforcement learning*, arXiv preprint arXiv:1611.01578, (2016).

Acrónimos

AutoML *Automated Machine Learning*.

CASH *Combined Algorithm Selection and Hyperparameter Optimization*.

HPO Optimización de hiperparámetros.

IA Inteligencia Artificial.

ML *Machine Learning*.

MLP *Multilayer Perceptron*.

PSO *Particle Swarm Optimization*.

SMBO *Sequential Model-Based Optimization*.

SVM *Support Vector Machine*.

Apéndices

A

Anexo del diseño técnico

En esta sección se explicarán más detenidamente los procesos B y C encargados del entrenamiento del clasificador de clasificadores y del combinador de clasificadores. Para ello se expondrán dos diagrama de flujo de datos de nivel 0.

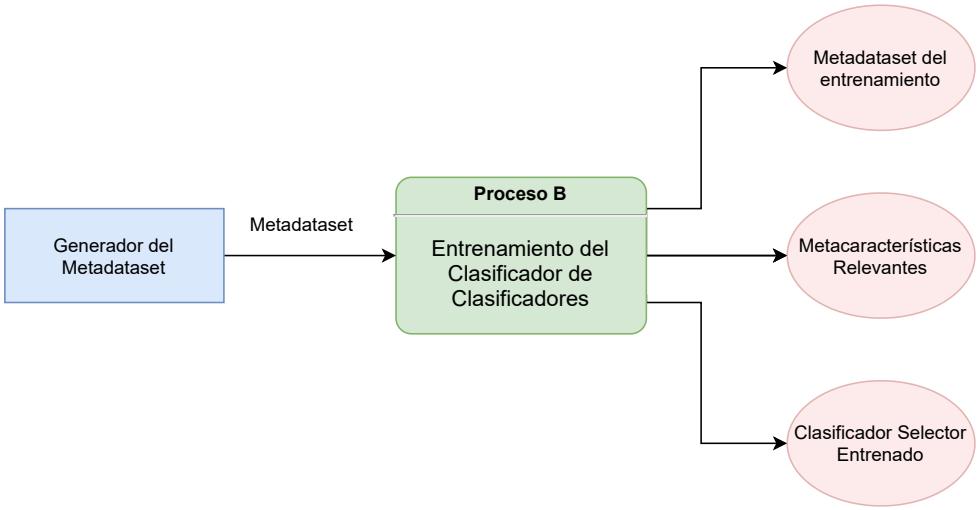

Figura A.1: Diagrama de flujo de datos de Nivel 0 del entrenamiento del Clasificador de Clasificadores

El **Proceso B** es el encargado del entrenamiento del clasificador de clasificadores. Este proceso recibe como entrada el metadataset para obtener los requisitos necesarios para realizar sus predicciones mediante el Proceso A.

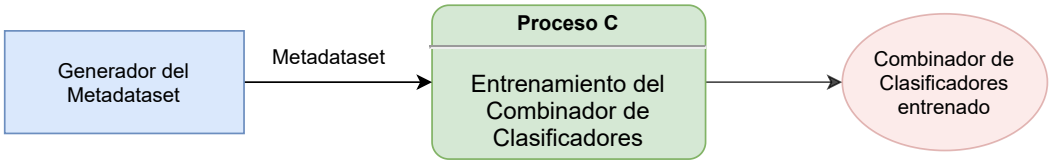

Figura A.2: Diagrama de flujo de datos de Nivel 0 del entrenamiento del Combinador de Clasificadores.

El **Proceso C** es el encargado del entrenamiento del combinador de clasificadores. Al igual que el

Proceso B, este proceso recibe como entrada el metadataset para obtener los requisitos necesarios para realizar sus predicciones mediante el Proceso D.

A continuación presentaremos los diagramas de flujo de datos de nivel 1 de ambos procesos en las siguientes figuras.

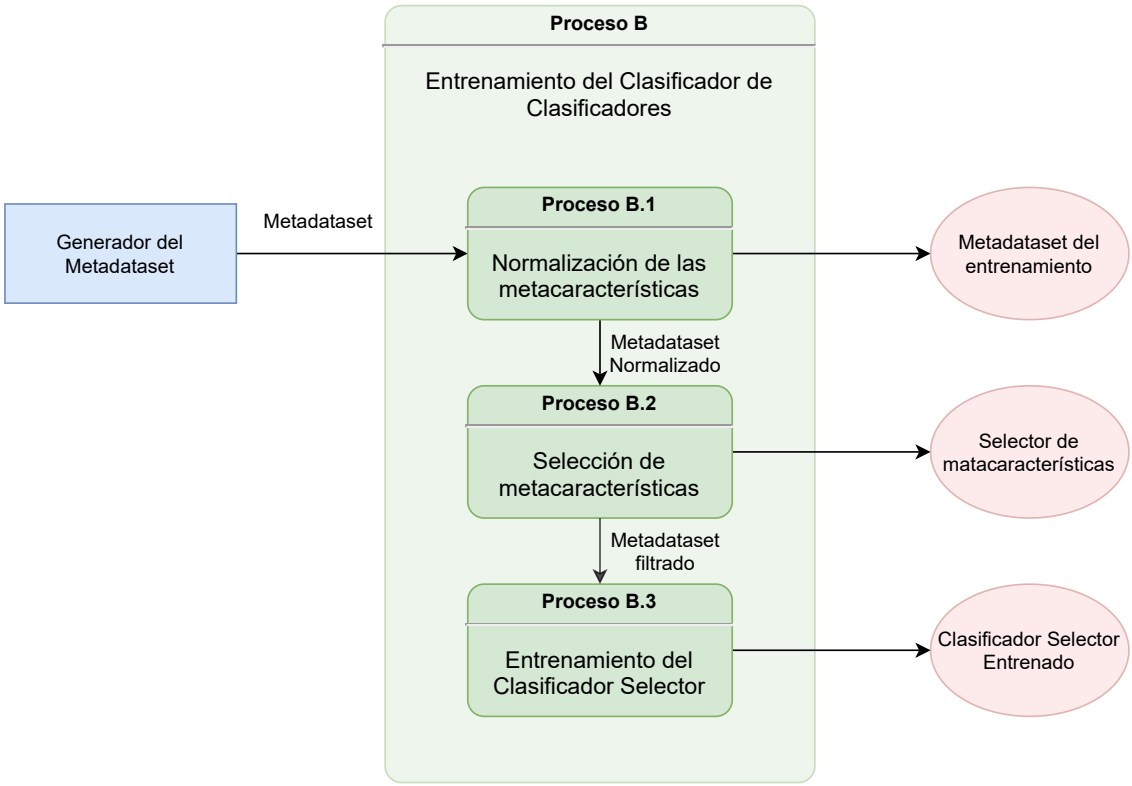

Figura A.3: Diagrama de flujo de datos de Nivel 1 del entrenamiento del Clasificador de Clasificadores.

El **Proceso B** está compuesto tres procesos internos los cuales son: **Proceso B.1**, **Proceso B.2** y **Proceso B.3**.

Este proceso recibe como entrada un metadataset cuyos datos serán normalizados (**Proceso B.1**) tal como se define en la subsección 4.1. De este proceso obtenemos el metadataset de entrenamiento que se utilizará en el Proceso A.2. Este metadataset normalizado nos servirá como entrada para el **Proceso B.2** donde se entrenará el selector de metacaracterísticas que se utilizará en el Proceso A.3. Tras finalizar este proceso, se obtendrá un metadataset que solo contendrá las metacaracterísticas más importantes. Este metadataset se utilizará para alimentar la entrada del **Proceso B.3** el cual será encargado de entrenar el clasificador selector que se utilizará para el **Proceso A.4**.

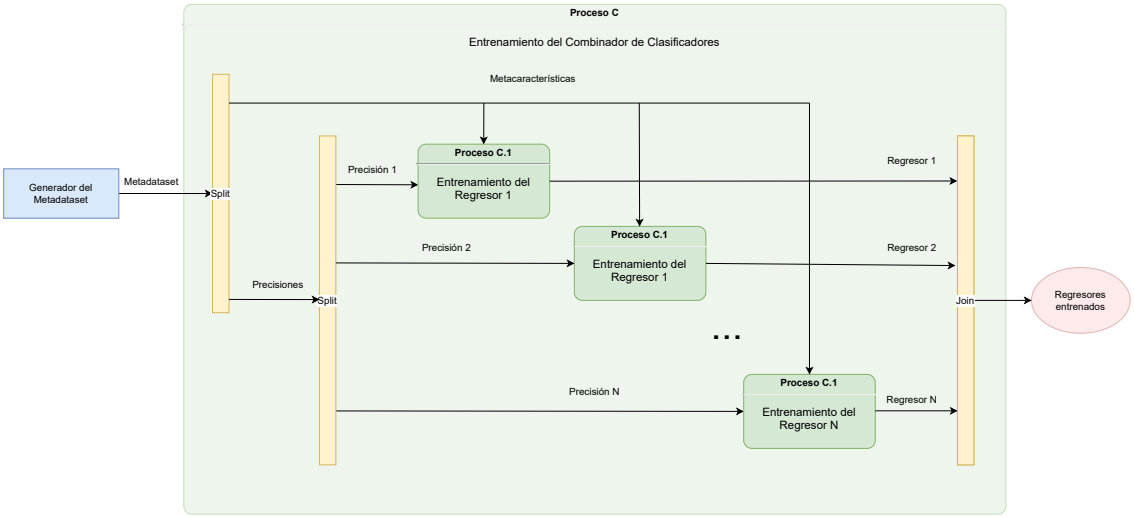

Figura A.4: Diagrama de flujo de datos de Nivel 1 del entrenamiento del Combinador de Clasificadores.

El **Proceso C** está compuesto únicamente por un proceso interno (**Proceso C.1**).

Para el entrenamiento del combinador de clasificadores, se ha dividido del metadataset, las predicciones de los casificadores candidatos. Por cada uno de los vectores de precisión obtenidos por las predicciones de los clasificadores candidatos, se ha entrenado un regresor (**Proceso C.1**) para estimar las predicciones que darían estos clasificadores. Estos regresores serán los que se utilizarán en el Proceso D.3 para obtener la política de importancias del combinador de clasificadores.

B

Datasets utilizados

En esta sección se mostrarán los datatasets que se han usado junto con la precisión media de algunos clasificadores tras la validación cruzada:

Dataset	Regresión Logística	SVM	MLP	Árbol de Decisión	Naive Bayes
acutel.data	100.00	58.33	65.83	100.00	95.83
acutel2.data	100.00	48.33	88.33	100.00	82.50
attr100.data	95.63	51.23	79.38	55.12	50.07
australian.data	86.09	73.77	76.52	81.30	86.38
autoUniv.data	73.10	76.40	73.20	66.90	71.80
bank-Marketing.data	88.74	88.45	87.33	86.80	83.68
breast-cancer.data	73.90	71.07	74.33	62.78	73.60
chscase-census6.data	58.25	57.25	56.00	51.50	56.75
chscase-funds.data	63.22	68.19	48.60	67.05	62.81
cloud.data	67.55	70.36	68.45	90.00	61.00
cmc.data	67.21	66.53	71.28	65.11	63.41
codebreaker.data	72.80	76.60	72.20	68.20	71.70
corral.data	90.63	100.00	100.00	100.00	86.25
cpu.data	95.21	93.31	77.45	97.62	94.76
delta-ailerons.data	64.86	94.11	93.87	91.11	93.32
disclosure-x-tampered.data	52.87	51.21	51.37	47.44	53.91
example1.data	42.50	85.25	85.50	82.50	40.50

Tabla B.1: Precisión media de algunos clasificadores tras la validación cruzada con los datasets extraidos.

Dataset	Regresión Logística	SVM	MLP	Árbol de Decisión	Naive Bayes
example3.data	48.25	96.25	99.00	98.50	46.25
FOREX-eurpln-day-Close.data	49.35	49.18	49.67	49.73	49.29
fri-c0-1000-5.data	84.30	90.60	91.20	82.50	87.80
german.data	74.60	70.80	68.20	69.60	73.50
ilpd.data	70.14	71.34	68.78	62.08	55.75
kin8nm.data	73.23	89.72	91.02	77.00	73.60
krvskp.data	95.93	97.53	99.34	99.69	62.52
mofn-3-7-10.data	100.00	100.00	100.00	100.00	87.15
mux6.data	60.26	100.00	100.00	100.00	57.95
parity5-plus-5.data	40.20	29.00	100.00	71.54	40.29
phoneme.data	74.96	84.66	86.73	86.66	76.09
piechart1.data	91.50	91.36	80.79	87.95	87.54
piechart2.data	96.92	97.72	92.58	96.25	93.56
piechart3.data	87.74	87.56	80.52	83.09	22.36
prnn-crabs.data	99.50	84.00	98.00	89.50	61.50
qsar-biodeg.data	86.73	82.19	87.20	82.09	72.32
qualitative-bankrupt.data	56.80	99.60	100.00	100.00	96.00
rmftsa-sleepdata.data	68.06	67.77	55.27	68.93	68.06
sa-heart.data	71.20	66.23	66.40	62.77	70.78
segment.data	99.74	95.89	99.70	99.35	82.94
sigma.data	74.95	84.44	87.14	87.29	76.00
simulation.data	91.30	91.48	91.48	89.44	88.15
space-ga.data	50.40	54.97	51.75	77.12	59.83
tae.data	64.96	65.67	66.96	80.13	72.92
thoracic-surgery.data	84.04	85.11	84.26	75.96	17.87
threeOf9.data	81.44	100.00	98.83	99.02	81.06
tic-tac-toe.data	69.41	89.56	87.89	89.04	71.71
tokyo1.data	79.88	90.62	83.32	90.62	88.12

Tabla B.2: Precisión media de algunos clasificadores tras la validación cruzada con los datasets extraidos.

Dataset	Regresión Logística	SVM	MLP	Árbol de Decisión	Naive Bayes
unknown.data	85.55	81.42	73.11	86.11	70.26
vertebraC.data	85.16	84.84	81.29	80.97	77.42
vowel.data	97.17	96.77	99.90	98.08	93.94
wilt.data	96.78	94.61	97.48	98.10	89.42
xd6.data	81.39	100.00	100.00	100.00	81.39

Tabla B.3: Precisión media de algunos clasificadores tras la validación cruzada con los datasets extraidos.

C

Anexo de los experimentos

En este anexo mostraremos los experimentos 1.1 y 1.2 de los datos sin normalizar junto con los resultados completos del experimento 2.1.

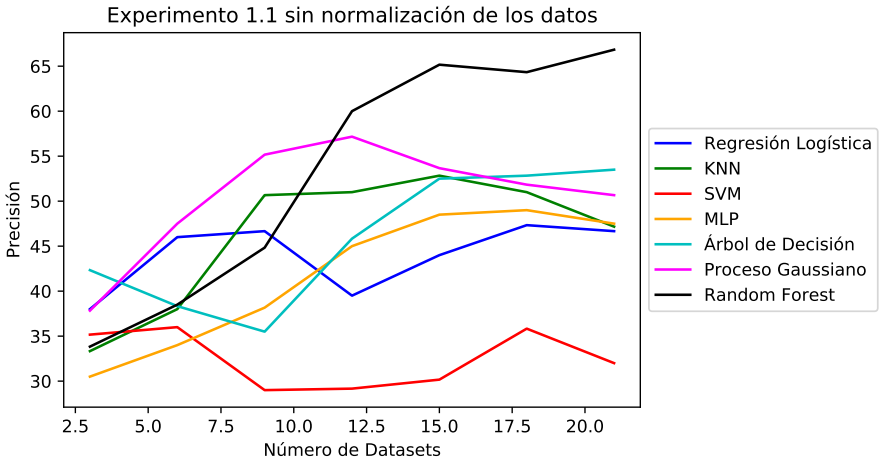

Figura C.1: Experimento 1: Relación entre el número de datasets en el entrenamiento y precisión de los clasificadores.

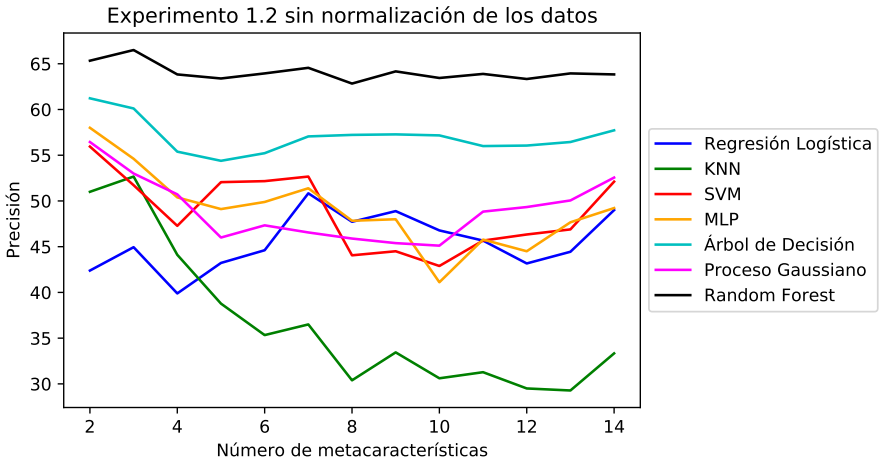

Figura C.2: Experimento 2: Relación entre el número de metacaracterísticas relevantes en el entrenamiento y la precisión suavizada de los clasificadores de clasificadores.

Dataset	SVM	MLP	Árbol de Decisión	Comb. Uniforme	Nuestro Comb.
acutel.data	58.33	65.83	100.00	80.83 +- 24.17	91.67 +- 17.08
attr100.data	51.23	79.38	55.12	68.4 +- 5.86	67.42 +- 6.28
australian.data	73.77	76.52	81.30	81.01 +- 3.46	82.03 +- 5.11
autoUniv.data	76.40	73.20	66.90	77.1 +- 4.41	77.4 +- 3.83
bank-Marketing.data	88.45	87.33	86.80	88.79 +- 1.22	88.61 +- 1.23
breast-cancer.data	71.07	74.33	62.78	74.06 +- 5.37	74.07 +- 7.15
chscase-census6.data	57.25	56.00	51.50	53.75 +- 9.03	57.0 +- 7.97
chscase-funds.data	68.19	48.60	67.05	65.53 +- 9.02	67.16 +- 11.11
cloud.data	70.36	68.45	90.00	69.36 +- 16.93	72.09 +- 17.95
cmc.data	66.53	71.28	65.11	72.23 +- 3.65	72.09 +- 3.27
codebreaker.data	76.60	72.20	68.20	75.9 +- 3.81	76.6 +- 4.88
cpu.data	93.31	77.45	97.62	94.29 +- 6.32	96.19 +- 5.55
fri-c0-1000-5.data	90.60	91.20	82.50	90.6 +- 3.35	90.6 +- 3.35
german.data	70.80	68.20	69.60	71.2 +- 3.74	70.2 +- 3.92
ilpd.data	71.34	68.78	62.08	70.66 +- 5.55	70.35 +- 6.47
kin8nm.data	89.72	91.02	77.00	90.31 +- 1.08	90.62 +- 0.95
parity5-plus-5.data	29.00	100.00	71.54	75.45 +- 8.6	75.45 +- 8.6
piechart3.data	87.56	80.52	83.09	85.88 +- 2.86	85.51 +- 3.24
prnn-crabs.data	84.00	98.00	89.50	95.0 +- 5.0	94.5 +- 5.68
qsar-biodeg.data	82.19	87.20	82.09	86.15 +- 2.21	85.68 +- 1.58
rmftsa-sleepdata.data	67.77	55.27	68.93	70.41 +- 4.57	70.22 +- 4.91
space-ga.data	54.97	51.75	77.12	68.62 +- 10.4	66.98 +- 10.91
tae.data	65.67	66.96	80.13	65.5 +- 10.38	66.88 +- 9.45
unknown.data	81.42	73.11	86.11	83.08 +- 6.96	84.13 +- 6.65
vertebraC.data	84.84	81.29	80.97	82.58 +- 6.64	82.58 +- 6.48
vowel.data	96.77	99.90	98.08	99.7 +- 0.46	99.49 +- 0.68
wilt.data	94.61	97.48	98.10	97.87 +- 0.93	97.71 +- 0.97

Tabla C.1: Resultados del experimento 2.1: Evaluaciones de nuestros combinadores de clasificadores junto con la evaluación de los clasificadores candidatos en los 27 datasets extraidos.